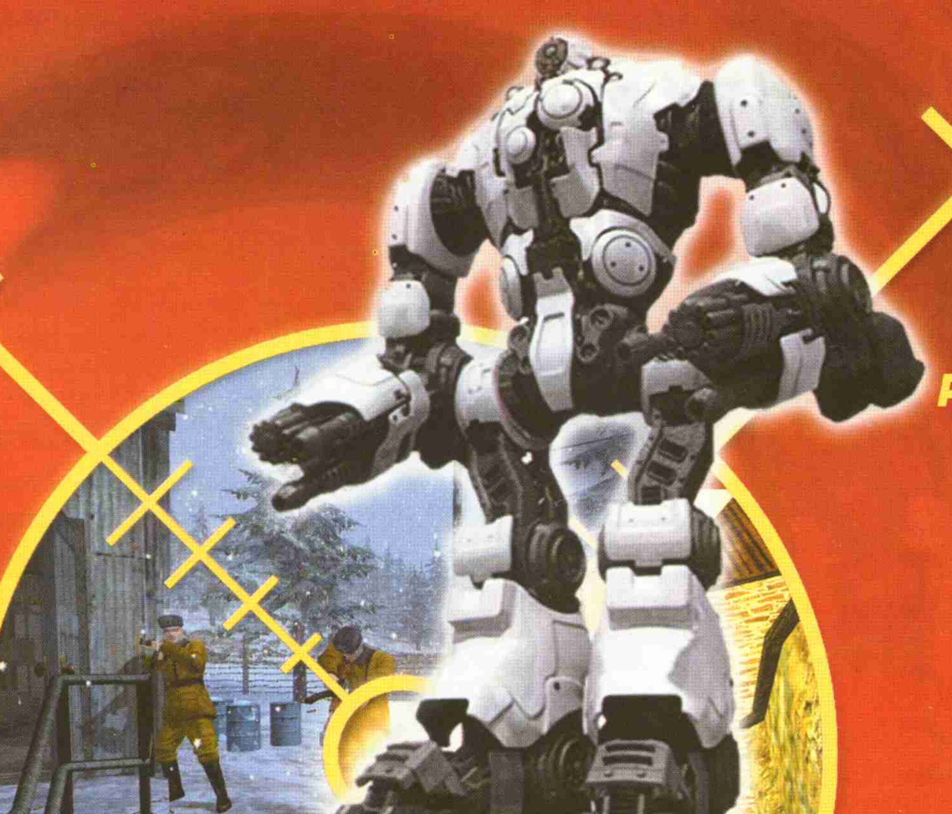
Videojuegos

Curso de Diseño y Programación

Nº 8 5,99 euros



Planificar el programa con pseudocódigo

Manejo de entidades con **Blitz3D**

Cómo **texturizar** las bionaves









Curso práctico de Videojuegos

AUTOR DE LA OBRA
Marcos Medina
DIRECCIÓN EDITORIAL
Eduardo Toribio
etoribio@iberprensa.com
COORDINACIÓN EDITORIAL
Eva-Margarita García
eva@iberprensa.com
DISEÑO Y MAQUETACIÓN
Antonio Gª Tomé
PRODUCCIÓN
Marisa Cogorro
SUSCRIPCIONES
Tel: 91 628 02 03
Fax: 91 628 09 35
suscripciones@iberprensa.com

FILMACIÓN: Fotpreim Duvial
IMPRESIÓN: Gráficas Don Bosco
DUPLICACIÓN CD.ROM: M.P.O.
DISTRIBUCIÓN
S.G.E.L.
Avda. Valdelaparra 29 (Pol. Ind.)
28108 Alcobendas (Madrid)
Tel.: 91 657 69 00

EDITA: Iberprensa
www.iberprensa.com


CONSEJERO
Carlos Peropadre

REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
C/ del Río Ter, 7 (Pol. Ind. "El Nogal")
28110 Algete (Madrid)
Tel.: 91 628 02 03
Fax: 91 628 09 35
(Añada 34 si llama desde fuera de España.)

DEPÓSITO LEGAL: M-35934-2002
ISBN: Coleccionable: 84 932417 2 5
Tomo 1: 84 932417 3 3
Obra Completa: 84 932417 5 X

NOTA IMPORTANTE:
Algunos programas incluidos en los CD de "Programación y Diseño de Videojuegos" son versiones completas, pero en otros casos se trata de versiones demo o trial, versiones de evaluación que Iberprensa quiere ofrecer a nuestros lectores. No se trata en ningún caso de las versiones comerciales de los programas, y las hemos incluido para dar al lector la oportunidad de conocer y probar esos programas y que así pueda decidir posteriormente si desea o no adquirir las versiones comerciales de cada uno.

Aprende divirtiéndote

Bienvenidos a **Programación y Diseño de Videojuegos**, la primera obra coleccionable cuyo objetivo es formar al alumno en las principales técnicas relacionadas en el desarrollo completo de un videojuego.

A lo largo de la obra el lector aprenderá programación a nivel general y a nivel específico con ciertas herramientas y lenguajes, aprenderá a trabajar con aplicaciones de retoque de imagen y también de diseño 3D y animación. Descubrirá las aplicaciones profesionales más importantes de audio y conocerá la historia de lo que se denomina "la industria del videojuego", los últimos 20 años, los juegos que marcaron un avance, sus creadores y en general la evolución del videojuego.

Pero además, esta obra tiene un segundo objetivo, desarrollar y potenciar la creatividad del lector, nosotros a lo largo de las diferentes entregas pondremos las bases y tú pondrás tu ingenio, tu creatividad y tu capacidad de mejorar.

Comienza aquí un viaje de 20 semanas articulado en 400 páginas y 20 CD-ROMs cuya finalidad es proporcionar las bases mínimas para después cada uno continuar su camino.

Recuerda que para alcanzar el éxito necesistas cumplir tres condiciones: que te gusten los juegos, poseer cierta dosis de creatividad y finalmente capacidad de estudio.

Una la cumples seguro.

sumario



PARA ENCUADERNAR LA OBRA:

- *Para encuadernar los dos volúmenes que componen la obra "Programación y Diseño de Videojuegos" se pondrán a la venta las tapas 1 y 2.*
- ***Tapas del volumen 1 ya a la venta.***
- ***Los suscriptores recibirán las tapas en su domicilio sin cargo alguno como obsequio de Iberprensa.***

SERVICIO TÉCNICO:
Para consultas, dudas técnicas y reclamaciones Iberprensa ofrece la siguiente dirección de correo electrónico: games@iberprensa.com

PETICIÓN DE NÚMEROS ATRASADOS:
El envío de números sueltos o atrasados se realizará contra reembolso del precio de venta al público más el coste de los gastos de envío. Pueden ser solicitados en el teléfono de atención al cliente 91 628 02 03

Codificación del juego, descripción y esquema del **programa**

Una vez completado el diseño de todos los elementos que componen el juego, el siguiente paso es implementarlos mediante programación para obtener un resultado tangible. Este proceso, por lo general, es muy laborioso y requiere de muchas pruebas. Un buen método para evitar el continuo ensayo-error es planificar todo el código mediante pseudocódigo. Pero antes, vamos a recordar algunos conceptos y premisas, necesarias poder acometer con más control nuestra labor.

(■) CONCEPTOS

Aunque la programación de ordenadores requiere el cumplimiento de ciertas reglas, generalmente impuestas por el lenguaje utilizado, constituye una disciplina con muchos matices artísticos y lógicos. Esta disciplina está repleta de algoritmos para resolver los diferentes problemas que plantea la obtención de un resultado preciso. Estos procesos están formados por un determinado conjunto de instrucciones y estructuras que, dependiendo de cómo se apliquen, se llegará al resultado final de una forma u otra. En las combinaciones que se puedan usar estará la clave para escribir un buen código que aproveche todos los gráficos y sonidos de forma aceptable para el desarrollo del juego. Hay que buscar siempre la máxima optimización para obtener el mayor rendimiento y que nuestro juego funcione aceptablemente en el mayor número de sistemas. Por lo general es imprescindible obtener rendimientos que no bajen de 30 frames por segundos (fps) en equipos medianos, para así evitar posibles saltos en el movimiento general del juego.

La modularidad, consistente en separar todo el código en pequeños programas, es también un buen método para organizarlo y estructurarlo mejor. Con este procedimiento resultará más fácil depurar errores (*bugs*) y, además, permite que varios programadores puedan trabajar en el mismo juego.

Otro concepto a tener en cuenta es procurar, lo máximo posible, el ahorro de memoria de sistema (RAM) y de vídeo, utilizando, por ejemplo, los tipos de variables adecuados o el tamaño de las texturas, controlando la creación y destrucción de entidades o borrando imágenes y audio en desuso.

Resumiendo, todas estas técnicas se aplicarán en la programación de "Zone of Fighters" de una forma u otra. Se van a utilizar unos métodos determinados, que llevarán a que el juego tenga una calidad aceptable.

(■) PREPARANDO Y ESTRUCTURANDO EL CÓDIGO

Vamos a basar la programación de "Zone of Fighters" en un sistema de modularidad. Para ello, dividiremos todo el código en pequeños programas (con extensión .bb), que luego serán incluidos desde el programa principal. Cada uno de estos programas se encargará de una tarea específica; de esta forma, podemos aislar los diferentes grupos de procedimientos dedicados a una labor en concreto.

```
MODULO "zone_of_fighters.bb"

Programa principal
    - Incluir módulos
    - Función Presentación
    - Función Definir modo gráfico
    - Función Cargar texturas y audio
    - Función Crear modelos
    - Función Crear emisores de partículas
      Bucle Principal
        - Inicializar número de luchadores
        - Función Menú
        - Si creamos un nuevo juego
          . Si el juego no es el primero
            Función Borrar entorno
            Función Borrar decorado
            Función Borrar enemigos
            Función Borrar mapa de colisiones
          . Función Crear entorno
          . Función Crear decorado
          . Función Crear OVNIS
          . Función Crear mapa de colisiones
        - Tocar música
        - Bucle de partida
            . Controlar FPS
            . Controlar tecla ESCAPE para salir
            . Función Controlar bionave
            . Función Actualizar entorno
            . Función Actualizar juego
            . Función Captura de la pantalla
            . Controlar la pausa
            . Actualizar mundo
            . Renderizar mundo
            . Función Indicadores de pantalla
            . Controlar visualización del panel ayuda. Tecla F5
        . Intercambiar los búferes
        - Repetir Bucle Principal hasta que se pulse ESCAPE
    Repetir Bucle Principal
Fin programa principal
```

1

Pseudocódigo del módulo principal "zone_of _fighters.bb".

A continuación se muestra una lista de los módulos y su cometido:

- **zone_of_fighters.bb:** Éste es el módulo principal del juego. Aquí se incluyen los demás módulos y es donde se encuentran el bucle principal y el bucle de cada partida.
- **definiciones.bb:** En este módulo se incluyen todas las definiciones de constantes, variables y estructuras que se utilizarán en el juego.
- **presentación.bb:** Aquí mostramos la presentación del juego. Pantallas del distribuidor, desarrollador y argumento.
- **menu.bb:** Este módulo contiene el menú principal y todos los submenús del juego.

- **funcarga.bb:** Aquí cargamos todas las texturas de los objetos del decorado, imágenes y todos los efectos de audio y voces.
- **funcpantaudio.bb:** Este módulo es el que contiene más funciones y se encarga de crear el entorno, los decorados, los modelos y el sistema de partículas. También controla los efectos especiales de audio y el modo gráfico.
- **decorado.bb:** Este módulo simplemente se encarga de colocar todo el decorado estático en el terreno.
- **jugador_principal.bb:** En este módulo, controlamos y actualizamos al jugador principal y todas sus características como el escudo, el camuflaje o los disparos.
- **jugadores_CPU.bb:** Aquí creamos y actualizamos a todos los OVNIS enemigos.
- **Fjuego.bb:** Este módulo contiene las funciones que manejan la cámara, el entorno, los indicadores, el mapa de colisiones y controla la evolución del juego.
- **enemigos.bb:** Con "enemigos" nos referimos a todas las plantas y animales que viven en el terrario. Así que este módulo los crea y controla. Además, también se encarga de los voladores o cubos kamicaces que pueblan la zona de combate.

Iremos por partes y explicaremos la estructura de cada módulo por separado mientras realizamos el pseudocódigo.

(■) MÓDULO "ZONE_OF_FIGHTERS.BB"

Éste es el programa principal. Realmente es pequeño ya que sólo incluye al bucle principal del juego, el cual contiene a su vez el bucle de cada partida. Al principio, incluimos todos los demás módulos por orden. Luego y después de la presentación, definimos el modo gráfico definitivo para el juego. Cargamos texturas y creamos modelos antes de entrar en el bucle principal. Una vez dentro de éste, llamamos al menú y, a continuación, creamos todos los elementos de la partida antes de introducirnos en el bucle de juego (Ver Fig. 1).

(■) MÓDULO "DEFINICIONES.BB"

No hay mucho que explicar aquí, sólo indicar los diferentes grupos de definiciones necesarios en el programa. No hay un orden específico a la hora de definir variables. Generalmente son las constantes las que encabezan la lista, así que empezaremos con ellas. Las utilizaremos para nombrar las entidades implicadas en el mapa de colisiones. Posteriormente, seguiremos con las variables que contienen las fuentes de letras y todas las matrices, variables y tipos que intervienen en el menú. A continuación, crearemos los *arrays* y estructuras necesarios para los bonos del juego, así como las utilizadas por los indicadores de pantalla.

Luego, definiremos todo lo necesario para el entorno del juego: zona de combate, variables de texturas del terreno, cielo, etc. Y para completar, las utilizadas por la cámara y las luces.

Seguimos con las variables globales utilizadas por el jugador principal y la estructura que define a los OVNIS. Para terminar con los jugadores, definimos la munición utilizada por ambos.

Le llega el turno al decorado, el cual necesita variables globales para el modelo y texturas y estructuras para controlarlos. De igual forma se define a los animales, plantas y voladores.

Terminamos las definiciones gráficas con las variables y tipos utilizadas para crear el sistema de partículas.

El módulo acaba con las variables referentes a todo el apartado sonoro del juego: música, efectos especiales de sonido (*sfx*) y voces (Ver Fig. 2).

```
MODULO "definiciones.bb"

 - Definir constantes para el mapa de colisiones
 - Definir fuentes de letras truetype que se utilizan para indicadores
 - Definir valores iniciales para el entorno
 - Matriz para los modos gráficos elegidos
 - Matriz para las elecciones en el  menú

 - Estructura para los botones (opciones) del menú
 - Matriz para el número de botones
 - Matriz para los títulos de las opciones (gráfico con la lista de opciones)
 - Variables de control del menú
 - Matriz para las diferentes texturas de los bonos
 - Matriz para el modelo de los bonos
 - Estructura para controlar los bonos
 - Definir variables para los indicadores de pantalla
 - Definir variables para la cámara, luces y ambiente
 - Definir variables para el control de la bionave (modelo, textura,
      pivote, sombra,coordenadas, velocidad, escudo, camuflaje,
      puntos, vida, munición y control para zurdos)
 - Definir estructura para el control de los OVNIS (modelo, pivote,
     sombra, coordenadas, velocidad, munición, distancia objetivo,
     vida)
 - Definir variables para munición de la bionave
 - Estructura del disparo (sprite, transparencia, alcance, luz, tipo de disparo)
 - Estructura de los agujeros y de las explosiones (sprite, luz)
 - Definir variables para munición de los OVNIS
 - Definir variables para los modelos del decorado, plantas y animales
 - Estructuras para los modelos inanimados (modelo y coordenadas)
 - Estructuras para los modelos animados (modelo, coordenadas, vida
     variables de la IA)
 - Variables y Estructura para los voladores
 - Variables para los 4 emisores de partículas
 - Estructura de los 4 emisores (posición, duración, dirección y velocidad
     del viento, velocidad partículas, velocidad emisión, tamaño del
     emisor, peso y elasticidad de las partículas, sprite, vida,
     transparencia, ángulo de rotación y tamaño)
 - Definir variables para el sonido (volúmenes, músicas, menú, explosiones,
     jugadores, plantas, animales y voces)
```

2

Pseudocódigo del módulo de definiciones "definiciones.bb".

(■) MÓDULO "PRESENTACIÓN.BB"

Se puede decir que es un módulo independiente del programa principal, ya que en él se define un modo gráfico y variables propias. Su sencilla labor es visualizar en pantalla los logos del distribuidor y del desarrollador, así como el argumento del juego (Ver Fig. 3).

(■) MÓDULO "MENU.BB"

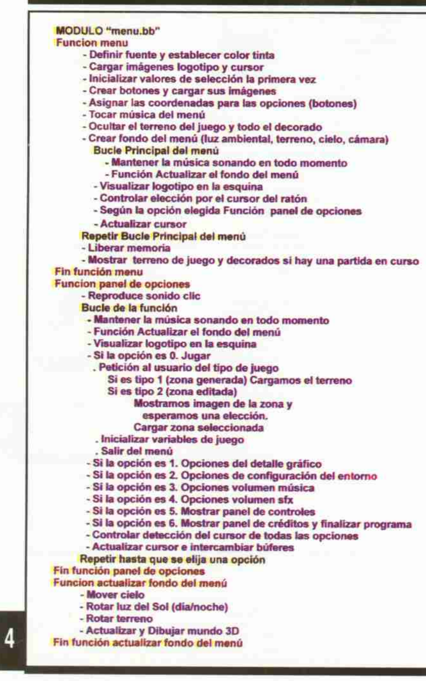
El menú es algo más complejo. Consta de un fondo con movimiento donde se muestra la vista de una cámara rotando en círculo

```
MODULO "presentacion.bb"

   - Definir modo gráfico
   - Cargar imágenes (distribuidor, desarrollador,
        logo del juego,  argumento)
   - Cargar voz del argumento
   - Mostrar Distribuidor
   - Función  Fade out Distribuidor
   - Mostrar Desarrollador
   - Función  Fade out Desarrollador
   - Reproducir voz con el argumento
   - Mostrar Logo del juego
   - Mostrar en scroll vertical imagen argumento
   - Función  Fade out Logo del juego
   - Liberar imágenes de la memoria
```

3

Pseudocódigo del módulo de presentación "presentacion.bb".



```
MODULO "menu.bb"
Funcion menu
    - Definir fuente y establecer color tinta
    - Cargar imágenes logotipo y cursor
    - Inicializar valores de selección la primera vez
    - Crear botones y cargar sus imágenes
    - Asignar las coordenadas para las opciones (botones)
    - Tocar música del menú
    - Ocultar el terreno del juego y todo el decorado
    - Crear fondo del menú (luz ambiental, terreno, cielo, cámara)
    Bucle Principal del menú
        - Mantener la música sonando en todo momento
        - Función Actualizar el fondo del menú
        - Visualizar logotipo en la esquina
        - Controlar elección por el cursor del ratón
        - Según la opción elegida Función panel de opciones
        - Actualizar cursor
    Repetir Bucle Principal del menú
    - Liberar memoria
    - Mostrar terreno de juego y decorados si hay una partida en curso
Fin función menu
Funcion panel de opciones
    - Reproduce sonido clic
    Bucle de la función
    - Mantener la música sonando en todo momento
    - Función Actualizar el fondo del menú
    - Visualizar logotipo en la esquina
    - Si la opción es 0. Jugar
        . Petición al usuario del tipo de juego
            Si es tipo 1 (zona generada) Cargamos el terreno
            Si es tipo 2 (zona editada)
                Mostramos imagen de la zona y
                esperamos una elección.
                Cargar zona seleccionada
        . Inicializar variables de juego
        . Salir del menú
    - Si la opción es 1. Opciones del detalle gráfico
    - Si la opción es 2. Opciones de configuración del entorno
    - Si la opción es 3. Opciones volumen música
    - Si la opción es 4. Opciones volumen sfx
    - Si la opción es 5. Mostrar panel de controles
    - Si la opción es 6. Mostrar panel de créditos y finalizar programa
    - Controlar detección del cursor de todas las opciones
    - Actualizar cursor e intercambiar búferes
    Repetir hasta que se elija una opción
Fin función panel de opciones
Funcion actualizar fondo del menú
    - Mover cielo
    - Rotar luz del Sol (día/noche)
    - Rotar terreno
    - Actualizar y Dibujar mundo 3D
Fin función actualizar fondo del menú
```

4

Pseudocódigo del módulo del menú "menu.bb".

```
MODULO "funcpantaudio.bb"

Funcion definir modo gráfico
    - Definir modo estándar
    - Visualizar pane-
Fin funcion crear entorno
Funcion borrar entorno
    - Borrar luces, terreno, agua, cámara y cielo
Fin funcion borrar entorno
Funcion borrar decorado
    - Borrar todos los edificios, árboles y rocas
    - Borrar todas plantas y animales
Fin funcion borrar decorado
Funcion borrar enemigos
    - Borrar todos los voladores
    - Borrar todos los OVNIS
Fin funcion borrar enemigos
Función definir modo gráfico
Imprimir lista de los drivers gráficos del sistema
    - Esperar elección del driver
    - Visualizar panel
    - Imprimir lista de modos gráficos
    - Esperar elección del modo gráfico
    - Esperar confirmación de elección
    - Definir modo gráfico
    - Activar doble búfer
Fin función definir modo gráfico
Funcion crear entorno
    - Activar o desactivar antialisado
    - Crear Sol y luz ambiental
    - Cargar el terreno correspondiente y aplicar texturas
    - Crear cielo y visualizarlo si corresponde
    - Crear agua y crear reflejo si corresponde
    - Crear cámara y definir rango de visión y zoom
    - Crear niebla si corresponde
- Posicionar la cámara
Funcion ocultar decorado
    - Ocultar todos los edificios, árboles y rocas
    - Ocultar todas plantas y animales
    - Ocultar OVNIS y voladores
Fin funcion ocultar decorado
Funcion mostrar decorado
    - Mostrar todos los edificios, árboles y rocas
    - Mostrar todas plantas y animales
    - Mostrar OVNIS y voladores
Fin funcion mostrar decorado
Funcion crear modelos
    - Cargar y posicionar bionave, crear pivote y sombra
    - Cargar y texturizar OVNIS,
    - Crear y texturizar voladores,
    - Crear y texturizar bonos
    - Cargar, texturizar y posicionar edificios, árboles y rocas
    - Cargar, texturizar y posicionar plantas y animales
    - Crear los sprites de los disparos y agujeros de impactos
```

6A

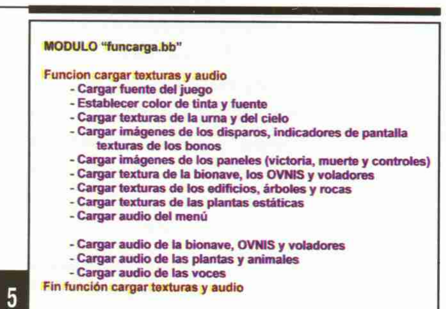
Pseudocódigo del módulo de funciones gráficas y de audio (I) "funcpantaudio.bb".

```
Funcion crear decorado
    - Colocar el decorado según el tipo de juego elegido
      . Si es tipo de juego 1 (Generada)
            Llamar a las funciones para colocar los
              edificios, rocas, árboles, animales y plantas
              según el terreno elegido
      . Si es tipo de juego 2 (Zona editada)
            Leer datos del fichero de la zona editada
            Bucle lectura del fichero
              Leer registro
              Colocar modelo según el valor del registro objeto$
             Repetir lectura del fichero hasta el último registro
            Cerrar fichero
Fin funcion crear decorado
Funcion crear emisores de particulas
    - Crear emisor 1
    - Crear emisor 2
    - Crear emisor 3
    - Crear emisor 4
Fin funcion crear emisores de particulas
Funcion  emisor de particulas 1
    - Manejar y visualizar cada particula
Fin funcion  emisor de particulas 1
Funcion  emisor de particulas 2
    - Manejar y visualizar cada particula
Fin funcion  emisor de particulas 2
Funcion  emisor de particulas 3
    - Manejar y visualizar cada particula
Fin funcion  emisor de particulas 3
Funcion  emisor de particulas 4
    - Manejar y visualizar cada particula
Fin funcion  emisor de particulas 4
Funcion  actualizar emisores
    - Mantener "vivos" los emisores 1 y 3
Fin funcion  actualizar emisores
Funcion  fade out
    - Se realiza un fundido a negro colocando un sprite
          delante de la cámara
Fin funcion  fade out
Funcion  salvapantalla
    - Al pulsar return salvamos el búfer de pantalla
Fin funcion salvapantalla
Funcion  Existe Fichero
    - Retorna true si el fichero existe y false si no
Fin funcion  existe fichero
Funcion  Play sonido
    - Asignar el volumen a los efectos de sonido según
          distancia de la cámara al emisor del sonido
    - Asignar el panorámico de los efectos de sonido según
          los valores de proyección de la cámara
          con respecto al emisor del sonido que se encuentre
          a la vista.
Fin funcion  Play sonido
Funcion  Radar
    - Visualizar los OVNIS, voladores y bionave con
          diferentes puntos de colores según
          su posición en el terreno
Fin funcion  radar
```

6B

Pseudocódigo del módulo de funciones gráficas y de audio (II) "funcpantaudio.bb".

sobre un terreno de combate. Sobre todo esto se imprimen una serie de apartados por donde el usuario navega para seleccionar las diferentes opciones del juego. Primero, se crean todos los botones que servirán de opciones, luego el fondo y a continuación se entra en el bucle principal del menú, en el cual se actúa dependiendo de la decisión del usuario tomada con el ratón. Para terminar el módulo, se crean las funciones que controlan las diferentes opciones y la actualización del fondo (Ver Fig. 4).



```
MODULO "funcarga.bb"

Funcion cargar texturas y audio
    - Cargar fuente del juego
    - Establecer color de tinta y fuente
    - Cargar texturas de la urna y del cielo
    - Cargar imágenes de los disparos, indicadores de pantalla
        texturas de los bonos
    - Cargar imágenes de los paneles (victoria, muerte y controles)
    - Cargar textura de la bionave, los OVNIS y voladores
    - Cargar texturas de los edificios, árboles y rocas
    - Cargar texturas de las plantas estáticas
    - Cargar audio del menú

    - Cargar audio de la bionave, OVNIS y voladores
    - Cargar audio de las plantas y animales
    - Cargar audio de las voces
Fin función cargar texturas y audio
```

5

Pseudocódigo del módulo de carga "funcarga.bb".

(■) MÓDULO "FUNCARGA.BB"

En este módulo simplemente cargamos todas las imágenes utilizadas por el sistema de partículas, indicadores y desarrollo del juego. También cargamos las texturas utilizadas por los modelos inanimados. Y para finalizar, todos los sonidos y voces (Ver Fig. 5).

(■) MÓDULO "FUNCPANTAUDIO.BB"

Este módulo engloba todas las funciones que permiten definir el modo gráfico, crear y actualizar el entorno, decorado, enemigos y sistema de partículas. También incluye un sistema especial de reproducción del audio, así como funciones de control de archivos (Ver Fig. 6A y 6B).

(■) MÓDULO "DECORADO.BB"

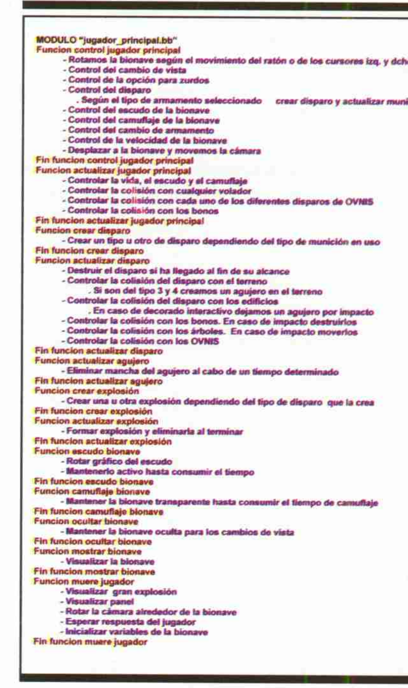
La finalidad de este pequeño módulo es la de crear y colocar cada uno de los elementos inanimados sobre el terreno (edificios, árboles, rocas, etc.) (Ver Fig. 7).

```
MODULO "decorado.bb"

Funcion colocar decorado
    - Crear y colocar cada uno de los edificios, rocas y árboles
      sobre el terreno aleatoriamente, dentro de unos valores
      Prefijados
Fin funcion colocar decorado
```

7

Pseudocódigo del módulo de decorados "decorado.bb".



```
MODULO "jugador_principal.bb"
Funcion control jugador principal
    - Rotamos la bionave según el movimiento del ratón o de los cursores izq. y dcho
    - Control del cambio de vista
    - Control de la opción para zurdos
    - Control del disparo
      . Según el tipo de armamento seleccionado    crear disparo y actualizar munición
    - Control del escudo de la bionave
    - Control del camuflaje de la bionave
    - Control del cambio de armamento
    - Control de la velocidad de la bionave
    - Desplazar a la bionave y movemos la cámara
Fin funcion control jugador principal
Funcion actualizar jugador principal
    - Controlar la vida, el escudo y el camuflaje
    - Controlar la colisión con cualquier volador
    - Controlar la colisión con cada uno de los diferentes disparos de OVNIS
    - Controlar la colisión con los bonos
Fin funcion actualizar jugador principal
Funcion crear disparo
    - Crear un tipo u otro de disparo dependiendo del tipo de munición en uso
Fin funcion crear disparo
Funcion actualizar disparo
    - Destruir el disparo si ha llegado al fin de su alcance
    - Controlar la colisión del disparo con el terreno
        . Si son del tipo 3 y 4 creamos un agujero en el terreno
    - Controlar la colisión del disparo con los edificios
        . En caso de decorado interactivo dejamos un agujero por impacto
    - Controlar la colisión con los bonos. En caso de impacto destruirlos
    - Controlar la colisión con los árboles.  En caso de impacto moverlos
    - Controlar la colisión con los OVNIS
Fin funcion actualizar disparo
Funcion actualizar agujero
    - Eliminar mancha del agujero al cabo de un tiempo determinado
Fin funcion actualizar agujero
Funcion crear explosión
    - Crear una u otra explosión dependiendo del tipo de disparo  que la crea
Fin funcion crear explosión
Funcion actualizar explosión
    - Formar explosión y eliminarla al terminar
Fin funcion actualizar explosión
Funcion escudo bionave
    - Rotar gráfico del escudo
    - Mantenerlo activo hasta consumir el tiempo
Fin funcion escudo bionave
Funcion camuflaje bionave
    - Mantener la bionave transparente hasta consumir el tiempo de camuflaje
Fin funcion camuflaje bionave
Funcion ocultar bionave
    - Mantener la bionave oculta para los cambios de vista
Fin funcion ocultar bionave
Funcion mostrar bionave
    - Visualizar la bionave
Fin funcion mostrar bionave
Funcion muere jugador
    - Visualizar  gran explosión
    - Visualizar panel
    - Rotar la cámara alrededor de la bionave
    - Esperar respuesta del jugador
    - Inicializar variables de la bionave
Fin funcion muere jugador
```

8

Pseudocódigo del módulo de control del jugador principal "jugador_principal.bb".

```
Funcion crear jugadores CPU (OVNIS)
    - Crear tantos OVNIS como indique
        el número de jugadores elegidos
Fin funcion crear jugadores CPU
Funcion actualizar jugadores CPU
    Por cada OVNI:
    - Controlar muerte del OVNI y recompensar al jugador
        con vida, puntos y munición
    - Controlar colisión del OVNI con los disparos de
        la bionave y de ellos mismos.
        En caso de impacto, actualizar vida y
        cambiar inmediatamente de dirección
    - Controlar la colisión con los bonos
    - Decrementar contador interno de acciones
    - Buscar a quién disparar
        el 50 % posibilidades busca otros OVNIS
        el otro 50 % busca bionave y calculamos la
        distancia al objetivo
    - Si está a cierta distancia del objetivo disparar
        . Dependiendo de la distancia, disparar un tipo
            de arma u otra
    - Desplazar OVNI hacia el nuevo destino
    - Al llegar a cero el contador interno, fijar las
        nuevas coordenadas de destino donde
        se encuentre la bionave.
        Inicializar de nuevo contador interno
    - Al llegar a la mitad del tiempo del contador interno, fijar nuevas
        coordenadas de destino aleatorias
    - Posicionar al OVNI (sombra, partículas)
Fin funcion actualizar jugadores CPU
```

9

Pseudocódigo del módulo de control de los OVNIS "jugadores_CPU.bb".

```
MODULO "Fjuego.bb"

Función mueve cámara
    - Establecer coordenadas de la cámara para el seguimiento
        de la bionave, aprovechando la posición y pivote de éste
    - Posicionar la cámara
Fin función mueve cámara
Función cambiar cámara
    - Según el tipo de cámara seleccionada ajustar zoom, distancia y
        velocidad de seguimiento
Fin función cambiar cámara
Función actualizar entorno
    - Mover el cielo si existe
    - Si corresponde, Crear día y noche rotando la luz del sol y
        disminuyendo la cantidad de luz ambiental
    - Si es terreno tipo 1, generar partículas para los volcanes
Fin función actualizar entorno
Función actualizar juego
    - Mantener la reproducción de la música
    - Controlar el final de partida en caso de eliminar a todos los OVNIS
    - Función actualizar jugador principal
    - Función actualizar jugadores_CPU
    - Función actualizar lunys
    - Función actualizar dreecks
    - Función actualizar slunks
    - Función actualizar shaarks
    - Función crear voladores hasta un máximo determinado
    - Función actualizar voladores
    - Función actualizar emisores partículas
    - Función crear bonos hasta un máximo de 10 continuamente
    - Función actualizar bonos
    - Determinar los tiempos de carga para cada munición
    - Función actualizar disparos
    - Función actualizar explosiones
    - Función actualizar agujeros si procede
Fin función actualizar juego
Función indicadores
    - Visualizar gráficos de los indicadores de pantalla
    - Visualizar datos en los indicadores de pantalla
Fin función indicadores
Función mapa de colisiones
    - Definir tipos de colisiones
    - Crear mapa de colisiones
Fin función mapa de colisiones
Función crea bonos
    - Crear n nuevo bono y posicionarlo aleatoriamente en el terreno
Fin función crea bonos
Función actualiza bonos
    - Rotar cada bono
    - Crear un emisor de partículas en cada bono
    - Controlar impacto con un disparo
Fin función actualizar bonos
Función final partida
    - Reproducir música
    - Mostrar panel de victoria hasta que se pulse disparo
    - Borrar música
Fin función final partida
Función pausa
    - Detener el juego en un bucle hasta que se pulse la tecla "p"
Fin función pausa
```

10

Pseudocódigo del módulo de control del juego "Fjuego.bb".

(■) MÓDULO "JUGADOR_PRINCIPAL.BB"

Este módulo es fundamental y es el encargado de controlar todo lo referente a la bionave de combate controlada por el jugador. Detecta las teclas y el ratón y actualiza el comportamiento de la bionave en el desarrollo del juego. Además, controla todo lo referente a los disparos y sus consecuencias así como el escudo y sistema de camuflaje. Para finalizar, se completa el módulo con la función encargada de la muerte del jugador (Ver Fig. 8).

(■) MÓDULO "JUGADORES_CPU.BB"

Otro módulo importante, encargado de gestionar todo lo referente a los OVNIS que controla el ordenador. Está compuesto por una función que los crea y otra que controla su IA (Inteligencia Artificial) (Ver Fig. 9).

(■) MÓDULO "FJUEGO.BB"

El desarrollo del juego es controlado aquí, en este módulo. Se maneja el movimiento de la cámara y los cambios de vista. Además, es aquí donde se le da vida al entorno creando el día y la noche o los cambios climatológicos. Otra función de este importante módulo es la actualización del desarrollo de la partida, es decir, la actualización de todas las acciones realizadas por el jugador, OVNIS, plantas, animales, voladores y bonos. Además, se controla la interactividad del decorado con la acción.

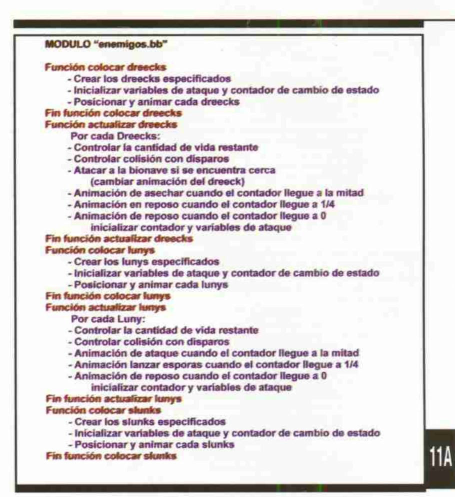
También aquí se actualizan los indicadores de pantalla y se crea el mapa de colisiones (Ver Fig. 10).

(■) MÓDULO "ENEMIGOS.BB"

Los enemigos son colocados y actualizados en este módulo, es decir, las plantas, animales y cubos voladores (Ver Fig. 11).

El uso de la modularidad implica la definición de más variables globales, ya que la mayoría son compartidas por multitud de funciones. Sin embargo, no es de alarmar la cantidad de memoria que este procedimiento pueda necesitar. A cambio, obtenemos beneficios a la hora de depurar, optimizar o ampliar el código.

Otra observación es que se ha evitado totalmente el uso de etiquetas (".label") en el juego (sólo se ha utilizado una etiqueta en la presentación), rechazando así instrucciones de cambio de flujo como "Goto" o "Gosub", principal enemigo de un programa estructurado.

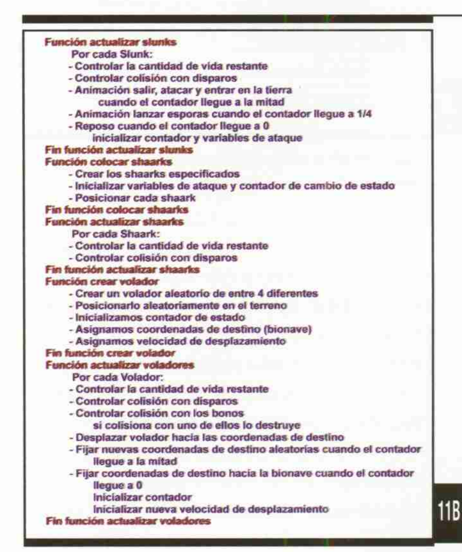


```
MODULO "enemigos.bb"

Función colocar dreecks
    - Crear los dreecks especificados
    - Inicializar variables de ataque y contador de cambio de estado
    - Posicionar y animar cada dreecks
Fin función colocar dreecks
Función actualizar dreecks
    Por cada Dreecks:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
    - Atacar a la bionave si se encuentra cerca
        (cambiar animación del dreeck)
    - Animación de asechar cuando el contador llegue a la mitad
    - Animación en reposo cuando el contador llegue a 1/4
    - Animación de reposo cuando el contador llegue a 0
        inicializar contador y variables de ataque
Fin función actualizar dreecks
Función colocar lunys
    - Crear los lunys especificados
    - Inicializar variables de ataque y contador de cambio de estado
    - Posicionar y animar cada lunys
Fin función colocar lunys
Función actualizar lunys
    Por cada Luny:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
    - Animación de ataque cuando el contador llegue a la mitad
    - Animación lanzar esporas cuando el contador llegue a 1/4
    - Animación de reposo cuando el contador llegue a 0
        inicializar contador y variables de ataque
Fin función actualizar lunys
Función colocar slunks
    - Crear los slunks especificados
    - Inicializar variables de ataque y contador de cambio de estado
    - Posicionar y animar cada slunks
Fin función colocar slunks
```

11A

Pseudocódigo del módulo "enemigos.bb".



```
Función actualizar slunks
    Por cada Slunk:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
    - Animación salir, atacar y entrar en la tierra
        cuando el contador llegue a la mitad
    - Animación lanzar esporas cuando el contador llegue a 1/4
    - Reposo cuando el contador llegue a 0
        inicializar contador y variables de ataque
Fin función actualizar slunks
Función colocar shaarks
    - Crear los shaarks especificados
    - Inicializar variables de ataque y contador de cambio de estado
    - Posicionar cada shaark
Fin función colocar shaarks
Función actualizar shaarks
    Por cada Shaark:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
Fin función actualizar shaarks
Función crear volador
    - Crear un volador aleatorio de entre 4 diferentes
    - Posicionarlo aleatoriamente en el terreno
    - Inicializamos contador de estado
    - Asignamos coordenadas de destino (bionave)
    - Asignamos velocidad de desplazamiento
Fin función crear volador
Función actualizar voladores
    Por cada Volador:
    - Controlar la cantidad de vida restante
    - Controlar colisión con disparos
    - Controlar colisión con los bonos
        si colisiona con uno de ellos lo destruye
    - Desplazar volador hacia las coordenadas de destino
    - Fijar nuevas coordenadas de destino aleatorias cuando el contador
        llegue a la mitad
    - Fijar coordenadas de destino hacia la bionave cuando el contador
        llegue a 0
        Inicializar contador
        Inicializar nueva velocidad de desplazamiento
Fin función actualizar voladores
```

11B

Más pseudocódigo de "enemigos.bb".

Durante la programación del juego haremos diferentes versiones de algoritmos, modificaremos posibles bugs o incluso añadiremos cosas nuevas. Este método servirá para aprender aún más y poder añadir aspectos personales y diferentes al juego.

⏭ En el próximo número...

... una vez estructurado todo el trabajo, empezaremos visualizando el terreno de juego.

Grafismo 2D. Texturizado de la bionave de combate

Un buen modelo no sirve de nada sin buenas texturas que lo puedan cubrir y le den el aspecto deseado.

Para texturizar nuestra bionave de combate, utilizaremos una imagen que servirá de piel (*skin*), la cual cubrirá el modelo. En esta imagen estarán dibujadas todas las diferentes partes del modelo. Sin embargo, es necesario que cada una de estas porciones del dibujo esté ubicada de tal forma que al estirarse sobre el objeto coincida en los polígonos correctos. Para poder hacer práctica esta ubicación es necesario disponer de una plantilla del mapeado UV.

Antes de nada, necesitaremos crear el *UV Mapping* del modelo. Esto es necesario para que los vértices de los polígonos se adapten al dibujo.

Actualmente, existen multitud de aplicaciones para obtener este tipo de mapeados, las cuales utilizan sistemas de proyección para generar las coordenadas de la textura en 2D a partir de los puntos 3D del modelo. Otra función de estos programas es que permiten mover estas coordenadas en el plano 2D para conseguir un ajuste más perfecto. Una vez creado el UV Mapping, éste queda guardado con el modelo y servirá posteriormente de referencia para que Blitz3D o cualquier aplicación de modelado adapte la textura correctamente. El procedimiento tradicional de dibujar sobre una plantilla lo aplicaremos en el texturizado de otros modelos del juego; para la bionave, vamos a utilizar una aplicación que permite dibujar directamente sobre el modelo 3D, Deep Paint 3D.

Sin embargo, para poder pintar directamente sobre nuestra bionave de combate sin problemas, necesitaremos también un mapeado UV de ella. La versión 2.0 de este programa, incluida en nuestro CD-ROM, incorpora un programa para realizar este tipo de mapeados por medio de distintos tipos de proyecciones llamado *MercatorUV*.



DEFINICIÓN

▸ TEMPLATE

Una plantilla o template es simplemente una representación gráfica de un UV mapping (ver Fig. 1).



DEFINICIÓN

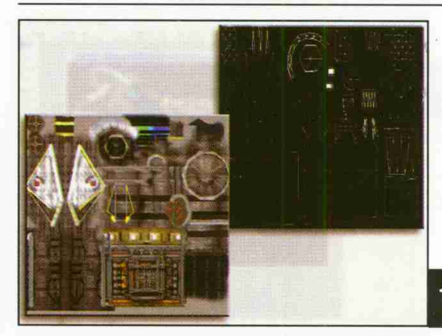
▸ UV MAPPING

UV Mapping o mapeado de coordenadas UV son posiciones colocadas en una imagen que se unen a puntos en un objeto 3D para ubicar una textura sobre él.

OPTIMIZAR EL MODELO

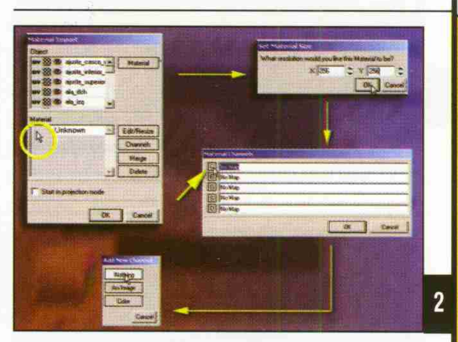
Antes de entrar en Deep Paint 3D, vamos a realizar una operación muy sencilla y útil que nos proporciona el programa Lithunwrap para optimizar nuestro modelo y así reducirlo de polígonos o vértices duplicados.

Dentro de Lithunwrap cargamos el modelo. Para optimizarlo elegimos la opción "Optimize models" situado en el menú "Tools". A continuación, con todas las opciones activadas pulsamos "Ok". Aparecerá una ventana de información, observamos como se ha conseguido reducir de polígonos la nave sin ninguna pérdida de calidad. Para terminar, salvamos de nuevo el modelo en formato .OBJ en la opción "Model" del menú "File".



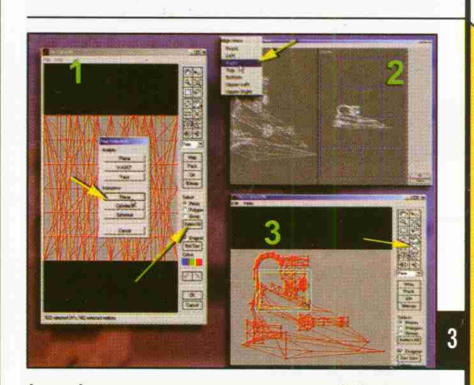
1

En la ilustración se muestra una imagen con toda la textura de la bionave y otra con la plantilla de despiece.



2

Los primeros cuatro pasos para empezar a trabajar con Deep Paint 3D.



3

Los primeros pasos para empezar a trabajar con MercatorUV.



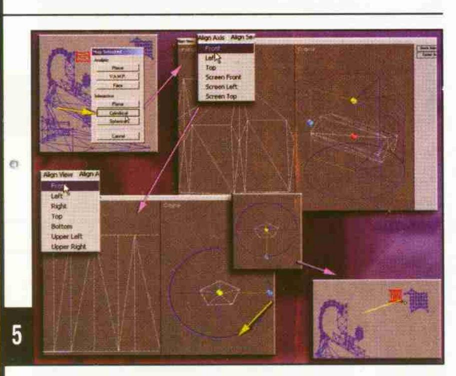
DEFINICIÓN

▸ TEXTURE MAPPING

El mapeado de texturas (texture mapping) consiste en adaptar una imagen 2D (textura) a un polígono 3D.



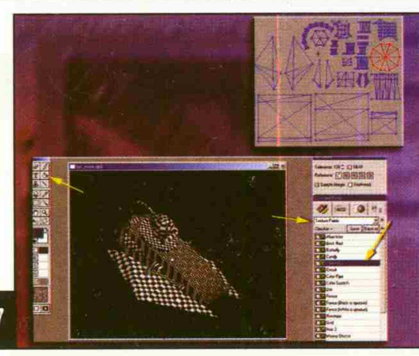
4 *Procedimiento para realizar el mapeado del casco de la bionave.*



5 *La cámara de la bionave necesita algunos ajustes para obtener el mapeado cilíndrico.*



6 *Con el ratón se puede mover directamente la pieza en la ventana "Unwrapped".*



7 *Una vez obtenido todo el UV Mapping, siempre es bueno realizar un chequeo.*

PRIMEROS PASOS CON DEEP PAINT 3D

Deep Paint 3D es una aplicación muy potente y cómoda de manejar. Su aspecto y la forma de utilizarlo se asemeja mucho al Adobe Photoshop. Iremos aprendiendo su uso a través de la práctica. A medida que trabajemos nuestros modelos descubriremos su funcionamiento fácilmente. Por lo tanto, vamos a proceder con el texturizado de la bionave. Empezamos cargando el modelo desde Deep Paint 3D en "File / Open". Aparecerá la ventana de diálogo "Material Import". Aquí, debemos crear un nuevo material para el modelo. Esto lo conseguimos pulsando al lado de "Unknow". En "Set Material Size" seleccionamos un tamaño para la textura de 256 x 256. Al pulsar "Ok" aparecerá "Material Channels", pulsamos en "C" para crear un canal de color y posteriormente en "Nothing" para tenerlo vacío (Fig. 2).

Ajustamos la vista pulsando dos veces en la herramienta lupa y rotamos en .

CREANDO EL UV MAPPING CON MERCATORUV

Como ya comentamos, debemos crear un mapeado de coordenadas UV antes de pintar sobre la bionave, para ello utilizaremos MercatorUV. Para entrar en esta aplicación pulsamos en el botón "Map" situado en la barra horizontal de herramientas. Una vez dentro del MercatorUV, debemos seleccionar todos los polígonos del modelo pulsando en "Select All". A continuación, crearemos una proyección lateral plana del modelo pulsando en el botón "Map" y eligiendo la opción "Planar" de "Interactive". Dentro de esta opción, elegimos la vista derecha "Right" en "Align View" (alinear vista) y "Ok". Con la herramienta "Scale" (escalar) reducimos la plantilla hasta que quede dentro del fondo gris (Fig. 3).

Debemos separar, manualmente, los grupos de polígonos que forman cada parte del modelo para realizar un mapeado individual. Para seleccionar cada grupo hay que activar la opción "Group" en "Select", luego con la herramienta de selección (M) elegimos un poco del casco. Al tener activada la opción "Group", MercatorUV automáticamente selecciona todo el casco, el cual se volverá de color rojo. Con la herramienta mover (T), lo separamos del modelo hacia cualquier parte del fondo gris. El siguiente paso será crear el mapeado del casco. Seleccionamos el mapeado interactivo esférico "Map / Interactive / Spherical". Se abre una ventana, en la que podemos ver cómo el casco se ha desenrollado (unwrapped) (Fig. 4).

Para el siguiente grupo, la cámara situada sobre el casco, elegiremos un mapeado cilíndrico "Map / Interactive / Cylindrical". Elegimos la vista frontal ("Align View / Front") y luego el alineamiento frontal de los ejes en "Align Axis / Front". A continuación, movemos la esfera azul (eje horizontal) de la ventana "original" hacia la parte inferior y pulsamos "Ok". Terminamos colocando el grupo junto al del casco (Fig. 5).

A continuación, se muestra una lista con el tipo de mapeado aplicado al resto de los grupos del modelo:

- **Conector al casco:** Map / Interactive Spherical ➡ Ok
- **Cuello y ajuste del cuello:** Map / Interactive Planar / Align View Top ➡ Ok
- **Tubo:** Map / Interactive Planar / Align View Right ➡ Ok
- **Ajuste tubo:** Map / Interactive Cylindrical ➡ Ajustes con las esferas amarilla y azul
- **Filtros, boquilla, escape y cubre escape:** Map / Interactive Cylindrical ➡ Ok



NOTA

En Deep Paint 3D podemos tener hasta cinco canales de efectos simultáneos distintos por cada material. "C" Color, "B" Bump Mapping (profundidad), "G" Glow (resplandor), "S" Shine (brillo) y "O" Opacity (opacidad).

Codificador: Map / Interactive Planar / Align View Top ➡ Movemos el objeto en la ventana "Unwrapped" hasta verlo desde arriba (Fig. 6).

Extremo cañón y cañón: Map / Interactive Planar / Align View Left ➡ Ok

Base del cañón, alas y alerones: Map / Interactive Planar / Align View Top ➡ Ok

Parte superior del casco: Map / Interactive Planar / Align View Top ➡ Ok

Parte inferior del casco y tapa del motor: Map / Interactive Planar / Align View Bottom ➡ Ok.

Estos dos últimos debemos reducirlos para que encajen dentro del fondo gris, con la herramienta de escalado.

Para finalizar, ajustamos todos los grupos con mucho cuidado dentro del fondo gris.

TEXTURIZANDO CADA GRUPO

Una vez creado el UV Mapping podemos pintar sobre la bionave directamente. Antes, es posible realizar un chequeo del mapeado, rellenando mediante la herramienta (K) con la textura "Checker+" de la lista "Texture Paints" ("Preset" F5) en el panel de comandos ("Command Panel"). Una vez comprobado, borramos la acción de relleno con "Undo" (CTRL+Z) (Fig. 7).

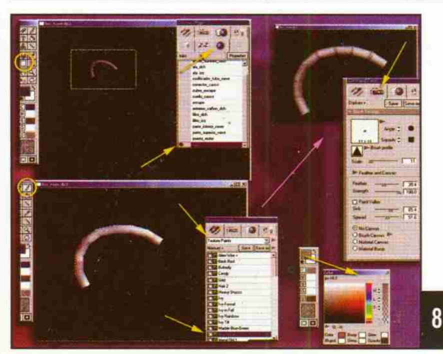
Vamos a comenzar dibujando el casco del protagonista, empezando por el tubo. Para ello, ocultaremos todos los demás elementos. En el panel de comandos ("Panel Command") pulsamos en "Element" (F7) y luego en "Objects". Aparecerá una lista con todos los grupos que forman la bionave. Ocultamos todos los grupos que no nos interesan pulsando en el "ojo" de cada capa. (Fig 069, 071). Una vez aislado el tubo, lo pintaremos con aspecto metálico. Elegimos la textura "Mercury" en "Preset / Texture Paints" y seleccionamos el tubo con la herramienta de selección (M). Hacemos un zoom hacia la pieza y con la herramienta de pincel, pintamos sobre la selección. Para perfilar las uniones entre segmentos, elegimos un pincel estándar y un color gris oscuro (Fig. 8).

Seguimos con los soportes del tubo. Borramos selección con CTRL+D, ocultamos el tubo y mostramos los soportes. Elegimos la textura "Metal Plate 7+" y pulsamos en "Brush and Paint Setting" (F6) para modificar la textura. Para obtener todas las opciones de edición, debemos pulsar sobre el símbolo "+" en "Texture Behavior" (comportamiento de la textura) y en "Advanced Behavior". Vamos a cambiar el tamaño de la textura moviendo totalmente hacia la izquierda el deslizador "ScaleX". Elegimos el pincel y pintamos los soportes. Para el conector del casco vamos a elegir la textura "Si-fi 3 +". En "Advanced Behavior" desplazamos el deslizador de tono "Hue" hacia -248 para cambiar el color y pintamos (Fig. 9).

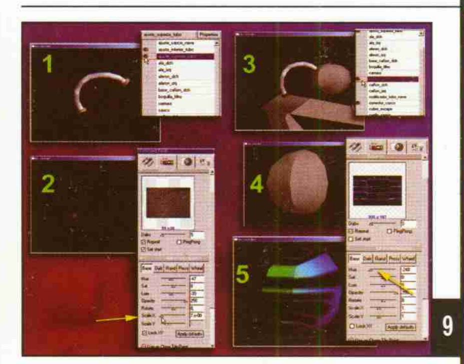
Para pintar el casco, lo aislamos y seleccionamos un color azul oscuro para luego pintar con el aerógrafo "AirBrush Sharp" en "Standard Tool" en el panel de comandos. Para dar la sensación de cristal, dibujaremos un brillo en el casco con "Airbrush Medium". Para realizar un pintado con más detalle podemos pasar al modo proyección pulsando en el icono (Fig. 10).

Siguiendo con la boquilla, la pintamos con la textura "Shift 3+" y la modificamos en tamaño, tono (Hue) y saturación (Saturation). Para los filtros, primero los pintamos de gris oscuro con una herramienta estándar ("Simple Flat") y luego con la textura "Alien Wire+" modificada en tamaño, color y luminosidad (Fig. 11).

Para el cuello y soporte del casco, elegimos la textura "Old metal 1". Una vez rellenas las piezas, vamos a proceder a pintar algunos detalles como radios y tornillos. Para dibujar los radios, mezclamos la visualización de la textura con los polígonos en wireframe con la opción "Display Wireframe", en la ventana emergente que aparece al pulsar el botón derecho del ratón sobre la vista. A continuación, seleccionamos el pincel "Gouache Bristle +"



Procedimientos para el pintado del tubo.



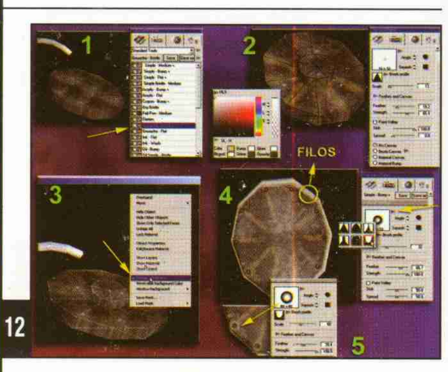
Cómo pintar los soportes del tubo y el conector del casco.



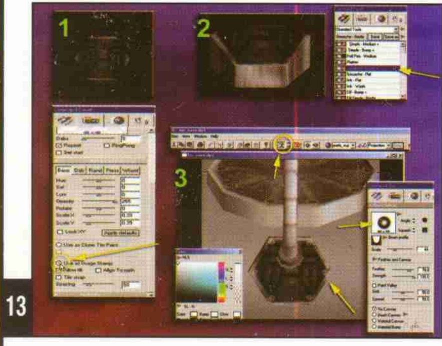
Procedimientos para pintar el cristal del casco.



Pasos para el pintado de la boquilla y los filtros.



Un trabajo especial de detalle requiere el cuello y su soporte.



Con la opción "Use as image stamp" colocaremos la textura como si se estampa un sello sobre el papel.



El modo de proyección nos ayudará a realizar un pintado con más detalle.



Procedimientos para el acabado de la estructura de la bionave.

y pintamos las líneas. Los tornillos los dibujamos en el modo de proyección. Tenemos que cambiar el tipo de pincel en "Brush Profile" y elegir la última opción, ajustamos la escala ("Scale") a 32, "Feather" a 66.7, Strength a 100.0 y pintamos los tornillos (Fig. 12).

Para rematar los filos y cantos de estas piezas, elegimos "Oil Bump" en "Standard Tools" y un color blanco. Para el codificador, elegimos la textura "Si-Fi 5". En "Advanced Behavior" activamos las opciones "Use as Image Stamp" y "Allow Tilt" para colocar la textura sin tilear. Ajustamos el tamaño "ScaleX y "ScaleY" a 0.39 y damos un sólo toque con el pincel en el centro de la pieza. Con "Gouache Bristle +" en "Standard Tools" perfilamos los bordes y el canto y, para los tornillos, cambiamos al modo proyección y pintamos con el mismo "Brush" que utilizamos en los tornillos anteriores. (Fig. 13).

Seguimos con la cámara. La aislamos del resto de piezas y seleccionamos su parte superior con la herramienta de selección (M) con "Face Selection Tools y "Feather" de 1. Pintamos con el pincel "Pastel Chalk" y un color casi negro. Con este mismo pincel dibujamos el frontal y la línea central de la parte superior. A esta línea le aplicamos el efecto "Emboss From Layer Below+" de la lista "Imagen Processing" con la herramienta de línea en el modo *proyección* (Fig. 14).

Para la parte superior de la nave, elegiremos la textura "Si-Fi 7 +", le reducimos la saturación de color, activamos de nuevo "Use as Image Stamp" y pinchamos en la pieza. Observamos cómo la textura cubre por completo la pieza. Si se coloca un poco desplazada podemos pinchar, un poco desplazado cada vez, en el objeto hasta que quede totalmente centrada. Empezamos detallando esta pieza, cambiando la tonalidad de la textura en aquellas partes que hagan contacto con otras piezas. Por ejemplo, el ajuste del cuello situado sobre la nave. Elegimos la opción "Darken" de la lista "Imagen Processing" y bordeamos con la herramienta pincel los contactos entre las piezas. Para la parte inferior, utilizaremos la textura "Si-Fi 5+" modificando "Sat" a -85 y "Lum" a 54 y de nuevo activamos "Use as Image Stamp". Para pintar los bordes laterales de las dos partes de la nave simularemos un material plateado con el pincel "Oil Bump+" y un color blanco (Fig. 15).

Pasamos a dibujar las alas y alerones, utilizaremos la textura "Si-Fi 5" con las siguientes modificaciones: "Hue" 0, "Sat" -255, "Lum" 55, "ScaleX 0.18,y "ScaleY" 1.0. Luego activamos "Use as Clone Tile Paint" en "Advanced Behavior" para hacer copias de la textura sobre la pieza con un ajuste tileado. Detallamos las alas con franjas naranjas en sus extremos que podemos realizar con la herramienta de línea. Aclaramos la textura del ala mediante un efecto "Lighten +" de la lista de "Imagen Processing" y pintando con un pincel estándar. La parte trasera de los alerones la pintamos con un color brillante y aplicamos un poco de "Glow" en "Imagen Processing" para simular luces encendidas (Fig. 16).

Para la tapa del motor aplicamos de nuevo la textura"Si-Fi 5 +" con los siguientes datos de modificación: "ScaleX" 0.23 y "ScaleY" 0.49. Y colocamos la textura con la misma opción "Use as Clone Tile Paint". En el escape y cubre escape utilizamos la misma textura "Si-Fi 5+". La base de los caños nes la cubrimos de un color oscuro y luego pintamos unas franjas anaranjadas. Para terminar el texturizado de nuestra nave quedan los cañones. Los pintamos con un color gris oscuro, luego con un proceso de oscurecimiento ("Darken +" en "Imagen Processing") dibujamos algunas líneas y manchas para dar el aspecto de que está manchado del humo de los disparos.

En el próximo número...

... añadiremos movimiento a nuestra bionave con CharacterFX.

Creando el sonido de los disparos, explosiones y **colisiones**

Ya estamos preparados para realizar los sonidos de "Zone of Fighters". Algunos de ellos partirán de otros sonidos procedentes de librerías de libre uso, los cuales modificaremos a nuestro gusto para obtener un nuevo sonido.

Generalmente, éste suele ser el procedimiento utilizado por la mayoría de las desarrolladoras, sobre todo si se trata de sonidos reales como ruidos específicos, sonidos de la naturaleza, etc. Sin embargo, también crearemos efectos sonoros desde cero, a partir de grabaciones desde micrófono o de partes extraídas de un CD. La idea básica que pretendemos es que el lector sepa sacar provecho de todas las posibilidades del editor de audio para crear sonidos nuevos a partir de otros. Antes de empezar debemos tener apuntado en papel una lista con los nombres que asignaremos a cada uno de los ficheros de audio que creemos para no despistarnos. "Zone of Fighters" utiliza cerca de 40 sonidos diferentes, así que explicaremos aquí los más significativos, suficientes para comprender los procedimientos para la creación de audio.

Todos los sonidos que fabriquemos los almacenaremos en el directorio "C:\juego_ZOF\Audio\". Toda esta organización de ficheros que estamos realizando hasta ahora nos servirá al final para crear la instalación del juego.

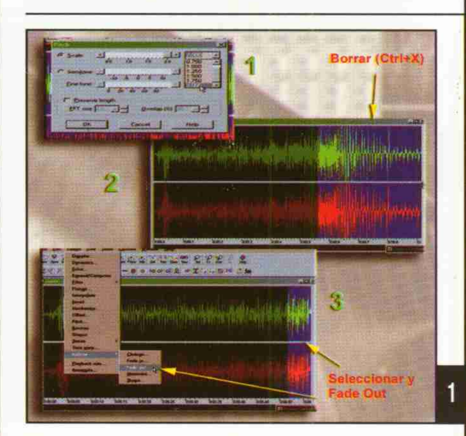
CREANDO LOS DISPAROS

Como sabemos, nuestra bionave de combate puede disparar distintos tipos de armas. Cada una sonará de una forma diferente al ser disparada, así que tendremos que realizar cuatro sonidos distintos. Además, los ovnis enemigos poseen tipos de armas análogas, por lo que reproducirán los mismos sonidos que la bionave. Todos los sonidos de los disparos los vamos a fabricar a partir de otros obtenidos de una librería. Solamente explicaremos los que necesiten operaciones diferentes.

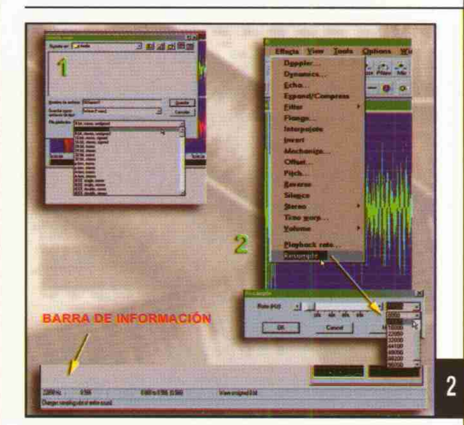
MUNICIÓN DE BAJO CALIBRE

Ejecutad el programa Goldwave y cargar el sonido "sonido1.wav" que se encuentra en el directorio "Extras" del CD que adjuntamos. En la reproducción podemos apreciar que se trata del sonido de una puerta automática. En primer lugar debemos establecer qué calidad vamos a utilizar en el juego. Para este tipo de efectos utilizaremos sonidos mono a 11025 Hz y 8 bits de calidad. Debemos recordar que antes de convertir a esta calidad es conveniente realizar todas las operaciones de edición. Así que empezaremos normalizando el volumen con la opción

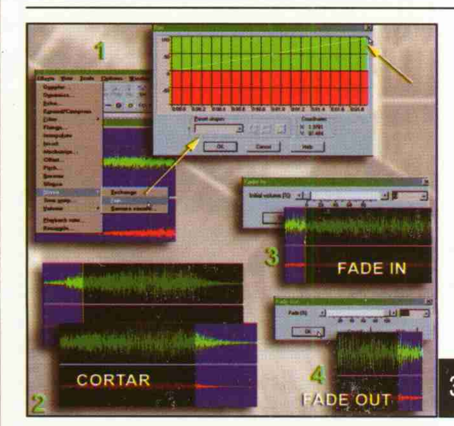
"Maximize" en "Effects \ Volume". Subimos a 1.5 en "New maximum". Al ser un disparo que se ejecutará rápidamente, necesitaremos un sonido corto y de ejecución rápida también. Así que tendremos que aumentar el tono aumentando la escala del "Pitch" también un punto más en "Pitch \ Effects" y subir a 2.000 en "Scale". Aun así comprobamos que sigue siendo algo largo, por lo tanto tendremos que cortarlo un poco. Para ir probando antes de cortar, iremos situando la línea de posición final y pulsamos "Play" para reproducir la selección. Una vez que hemos elegido la porción que nos interesa situamos las líneas de posición englobando el trozo que queremos borrar y pul-



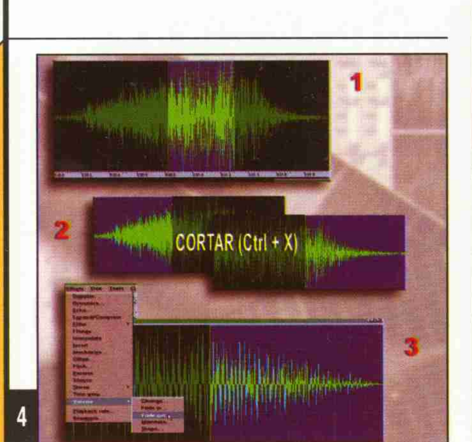
Gracias a la opción fade, lograremos un buen acabado.



Si resampleamos de nuevo, obtendremos la calidad deseada.



Crearemos el sonido de los misiles.





Crearemos las bombas de retardo.

samos CTRL + X. Volvemos a situar las líneas de posición englobando todo el sonido o CTRL + A para seleccionarlo completo y probamos el resultado. Para lograr un buen acabado, haremos un "Fade out" al final para "matar" el sonido. Seleccionamos un trozo del final y aplicamos el fade en "Effects \ Volume \ Fade out" con valor 100 (Fig. 1).

Una vez hayamos terminado, debemos pasarlo a la calidad que vayamos a utilizar. Para ello salvamos el sonido en "Save as" y en la opción "File Attributes" elegimos "8 bits, mono, unsigned", damos un nombre al archivo (nosotros hemos utilizado una "S" de "sonido" delante para identificar al fichero), "Sdisparo1" y guardamos en el directorio establecido "C:\juego_ZOF\Audio\". Al salvar el programa nos preguntará si queremos actualizar el sonido que estamos editando por el nuevo sonido cambiado de formato. Al decirle que "Sí" el sonido que estábamos editando cambia. En la barra de inferior de información observamos que todavía sigue a calidad 22025 Hz. Para pasarlo a la calidad deseada, es decir 11025 Hz debemos "Resamplearlo" de nuevo con la opción "Effects \ Resample" y eligiendo el valor 11025 (Fig. 2).

Para salvar es suficiente con pulsar CTRL + S.

MISILES

Para los misiles partiremos de un sonido de avión. Cargamos "sonido2.wav" del CD. Aquí nos encontramos con un problemilla. Observamos que está en estéreo y además el panorámico cambia de izquierda a derecha. No debe de ser un problema, porque al pasar a mono unificamos los dos canales, pero seguramente perderemos algo del efecto que buscamos. Así que, para evitar cualquier contratiempo, vamos a modificar el panorámico de nuevo. Elegimos la opción "Effects \ Stereo \ Pan" y subimos al máximo el punto final de la gráfica. A continuación, normalizamos y borramos los extremos como se muestra en la figura 3

Realizamos un "Fade in" (a 0) al principio y un "Fade out" (a 100) al final y salvamos con la calidad 8 bits, mono unsigned con el nombre "SDisparo2".

BOMBAS DE RETARDO

Para construir este sonido cargamos "sonido3.wav" del CD. Se trata del sonido de un disparo cualquiera, pero nosotros lo cambiaremos hasta obtener el que deseamos. Primero vamos a ecualizar para obtener más graves. Entramos en la opción "Effects \ Filtres \ Equalizer" y subimos en 24 Db la frecuencia de 60 (dos veces 12 Db) y bajamos en 12 la de 6000 Hz. Luego modificamos la escala del tono a 0.75. Continuamos dándole la vuelta con "Reverse" y añadiendo un silencio al principio, según las marcas de selección y eligiendo "Effects \ Silence". (Fig. 4).

CREANDO LAS EXPLOSIONES

Realmente, las explosiones se pueden crear de muchas formas, desde grabar una onomatopeya a través del micrófono, modificando un sonido ya existente o a partir de un ruido generado como el viento. Nosotros vamos a crear una explosión general, la cual nos servirá para aprender la técnica para cualquier tipo de explosión. Partiremos del sonido anterior "sonido2.wav". Una vez cargado, lo convertimos a mono como explicamos anteriormente (podemos utilizar cualquier nombre de archivo, por ejemplo, "explosion". Normalizamos y aislamos la parte central eliminando los extremos. A continuación, simplemente bajamos el tono en 0.5. Para terminar tendremos que ajustar el final con un "Fade out".

NOTA

▸ SONIDOS DE COLISIONES

Para crear el sonido de un impacto metálico similar al que puedan sufrir la bionave o los ovnis, se podrían utilizar efectos de reverberación o mecanizado a un sonido de golpe seco (sonido4.wav del CD) Sin embargo, la mejor forma es la tradicional, es decir, grabar a través del micrófono un golpe a algo metálico como un utensilio de cocina o un trozo de chapa. También podemos utilizar el sonido de los metales de una percusión de un teclado externo.

En resumen, para obtener el sonido deseado es sólo cuestión de tener clara la idea de qué es realmente lo que se quiere. Los sonidos procedentes de librerías de libre uso constituyen una buena base para experimentar con todas las posibilidades del editor de sonido. Hemos podido comprobar cómo, modificándolos adecuadamente, se pueden obtener sonidos realmente diferentes y acordes con nuestras necesidades.

En el próximo número...

... explicaremos las pautas a seguir para crear sonidos de ambiente y el tratamiento adecuado de la voz para conseguir la "voz en off" de nuestro juego.

Manejo de entidades

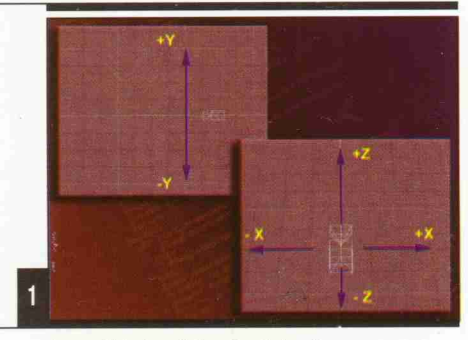
Para Blitz3D, todo lo que se desenvuelve dentro de un mundo 3D se denomina "Entidad". Los sprites que estudiamos en la entrega anterior forman parte de estas entidades.

Pero además, hay otros elementos que pertenecen a este concepto, como son: cámaras, luces, objetos 3D, planos, terrenos y pivotes. Cada una de estas entidades tiene una labor diferente dentro de un programa, sin embargo, todas poseen algo en común: una posición, una rotación y un tamaño determinado. Esto significa que pueden ser movidas, rotadas o escaladas. Además, todas las entidades pueden tener una entidad "madre" a la que estarán unidas. Si la "madre" se mueve, rota o escala, la entidad "hijo" también lo hará.



NOTA

Las entidades se posicionan utilizando el sistema de coordenadas X, Y, Z. El centro del mundo 3D es la posición 0, 0, 0, que es donde se colocan todas las entidades por defecto, incluida la cámara.



Las entidades vistas desde la cámara se mueven en los ejes x, y y z.

No abordaremos, de momento, todas las funciones de manejo de entidades en este número, pero sí las más importantes. De todas formas, durante el estudio de las distintas entidades, encontraremos nuevas funciones para aprender.

MOVIENDO ENTIDADES

Encontramos algunas funciones para desplazar, rotar y escalar las entidades. La primera función que estudiaremos será cómo posicionar una entidad en el mundo 3D.

```
PositionEntity entidad,
coordenada X#, Y#, Z#
```

Con esta función, la entidad también se puede desplazar cambiando algunas de las coordenadas, pero lo hará siempre de una manera relativa con respecto a la posición y orientación de la cámara.

Para mover una entidad dependiendo de su posición y orientación utilizaremos:

```
MoveEntity entidad,
coordenada X#, Y#, Z#
```

Si por el contrario sólo queremos que dependa de su posición utilizaremos:

```
TranslateEntity entidad,
coordenada X#, Y#, Z#
```

Para comprender mejor estas funciones, vamos a adquirir una entidad de tipo *mesh*, es decir, un objeto 3D, por ejemplo, un cilindro, el cual utilizaremos en los ejemplos. En "ejemplo8_1.bb" se puede ver el funcionamiento de estas tres funciones realizando las mismas operaciones (Fig. 2).



Hay varias funciones para mover entidades.

Tenemos también dos formas de rotar una entidad: *RotateEntity* y *TurnEntity*:

```
RotateEntity entidad, pitch#,
yaw#, roll#
TurnEntity entidad, pitch#,
yaw#, roll#
Pitch es referente al eje X,
yaw al Y y roll al eje Z.
Repeat
...
  RotateEntity cilindro1,30,0,0
  TurnEntity cilindro2,30,0,0
...
Forever
```

En este bucle, se rota el "cilindro1" sólo 30 grados en su eje X (orientación absoluta), mientras que el "cilindro2" rotará conti-



NOTA

Es importante recordar siempre que si la cámara se mueve de derecha a izquierda lo estará haciendo en el eje X (dcha +X / izq –X), si sube o baja, en el eje Y (subir +Y / bajar –Y) y si avanza o retrocede, en el eje Z (avanzar +Z / retroceder –Z). Lo mismo ocurre con las entidades vistas desde la cámara (Ver Fig. 1).

```
Graphics3D 800,600,16,1
SetBuffer BackBuffer()
Camara=CreateCamera()
Light=CreateLight()
Cono=CreateCone(32)
PositionEntity Cono,-1,0,3
Cono2=CreateCone(32)
PositionEntity Cono2,1,0,3
Repeat
 If KeyDown(200) pitch#=pitch#+1
 If KeyDown(203) yaw#=yaw#-1
 If KeyDown(205) yaw#=yaw#+1
 If KeyDown(16)  roll#=roll#-1
 If KeyDown(17)  roll#=roll#+1
 RotateEntity Cono,pitch,yaw,roll
 TurnEntity Cono2,10,0,0
   RenderWorld
   Flip
Until KeyDown(1)
```

3

Los dos cilindros tienen distintos grados de rotación.

nuamente de 30 en 30 grados en su eje X (posición relativa a su orientación) (Ver ejemplo8_2.bb) (Fig. 3).

Con PointEntity, podemos colocar una entidad en el sitio de otra. Su estructura es la siguiente:

```
PointEntity entidad,
entidad de destino
```

También podemos alinear los ejes de una entidad a un vector determinado, es decir, con esta función podemos hacer que una entidad gire en busca de otra. Un buen ejemplo de este tipo de funcionamiento es el seguimiento que un cañón situado en una torreta hace al objetivo (Ver ejemplo8_3.bb) (Fig. 4).

Utilizaremos la función:

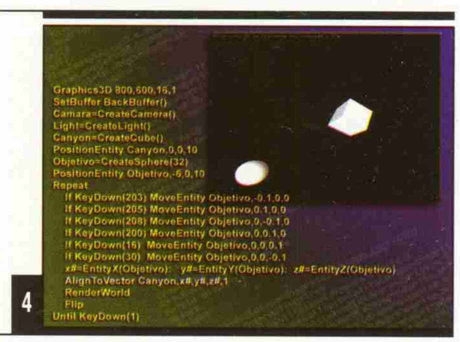
```
AlignToVector entidad, vector
X#, vector Y#, vector Z#, eje
[,proporción]
```

El "eje" de la "entidad" se alineará al "vector X#, vector Y#, vector Z#" con una transición de alineamiento según "proporción" (para Blitz3D superior a la versión 1.66).

DEFINICIÓN

▸ VECTOR Y NORMAL

En un espacio 3D, un *vector* es un segmento de recta colocado en cualquier punto y con una dirección determinada. Una *normal* es un vector perpendicular a cualquier polígono.



Ejemplo de una entidad que gira en busca de otra.

CONTROLANDO ENTIDADES: CONTROL BÁSICO

En toda manipulación de entidades es fundamental disponer de funciones para su control. Podemos empezar con las que controla su visibilidad. Para ocultar una entidad y que además no pueda ser detectada utilizaremos *HideEntity entidad*. Si por el contrario queremos detectarla pero no verla, la mejor forma es volverla transparente poniendo a 0 su canal alfa con *EntityAlpha entidad,0*.

Para volver a mostrarla utilizamos *ShowEntity entidad* o *EntityAlpha entidad,1*.

Una función muy interesante y utilizada es *CopyEntity entidad [,pariente]*. Permite copiar una entidad en otra, es decir, crear un clon. Muy útil en la creación de tipos de entidades.

Si en un programa dejamos de utilizar una entidad es fundamental borrarla para salvar memoria, para ello utilizaremos la función *FreeEntity entidad* (Ver ejemplo8_4.bb) (Fig. 5).

CONTROLANDO ENTIDADES: EFECTOS ESPECIALES Y OTROS

A las entidades se les pueden aplicar algunos tipos de efectos como color, transparencias, brillo especular, mezclas, vértices coloreados, etc.

Por ejemplo, *EntityColor entidad, rojo#, verde#, azul#* proporciona un color a una entidad.

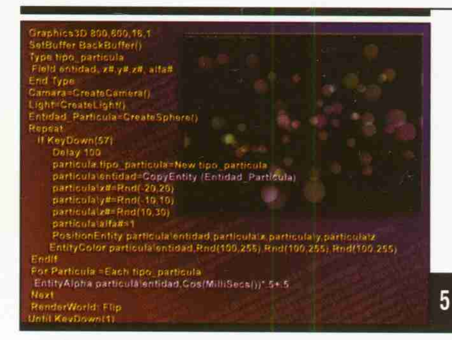
Para conseguir un brillo especular en un objeto, es decir, que se le iluminen ciertas áreas cuando recibe una luz directa, utilizaremos la función *EntityShininess entidad, brillo#*. El rango de "brillo" va desde 0 hasta 1. Así que un brillo medio sería 0.5.

También podemos realizar efectos de mezcla entre varias texturas aplicadas a un objeto con la función *EntityBlend entidad, tipo de mezcla*.

En un próximo número estudiaremos con más detalle este tipo de función cuando aprendamos el poder de Blitz3D para texturizar objetos. Pero podemos anticipar ya la función que permite realizar esta acción mágica, se trata de *EntityTexture entidad, textura [,fotograma][,indice]*.

Aquí, "textura" es la variable que contiene la textura cargada de disco y "fotograma" (frame) especifica qué fotograma de la textura animada se utilizará. Blitz3D permite mezclar varias texturas en el mismo objeto -este proceso se denomina "Multitexturing", por ejemplo, lo utilizamos en el texturizado del terreno de combate. De tal forma que "indice" indica el número de la textura asignada al objeto (de 0 – 7).

Pero hay una función especialmente preparada para ofrecer algunos efectos adicionales a una



Con FreeEntity, borraremos una entidad.



NOTA

Es importante saber que las funciones de visibilidad aplicadas a una entidad afectarán también a sus entidades hijos si las tuviera.

entidad, nos referimos a *EntityFX entidad, efecto*. Hay varios tipos de efectos, desde aplicar brillo a todo el objeto hasta mostrar, en modo wireframe (alámbrico), todos los polígonos.

Para obtener la madre de una entidad se utiliza la función *GetParent Entidad* y para unirla a otra utilizaremos *EntityParent entidad hijo, entidad madre*.

EL ESTADO DE LAS ENTIDADES. OBTENIENDO INFORMACIÓN

En este apartado encontramos funciones muy potentes que nos permitirán obtener información muy útil de cada entidad. Vamos a comenzar con unas funciones básicas, las cuales nos proporcionarán en cualquier momento la situación de una entidad en el mundo 3D a través de sus coordenadas.

```
EntityX# (entidad) EntityY#
(entidad) EntityZ# (entidad)
...
PositionEntity cubo,0,3.5,-10
PosX#=EntityX(cubo):
PosY#=EntityY(cubo):
PosZ#=EntityZ(cubo)
Text 10,10,"La posición X del
cubo es: " + PosX ;← PosX= 0
Text 10,30,"La posición Y del
cubo es: " + PosY ; ← PosY=3.5
Text 10,60,"La posición Z del
cubo es: " + PosZ ; ← PosZ= -10
...
```

De igual forma podemos obtener los ángulos X, Y y Z de una entidad a través de las funciones:

```
EntityPitch# (entidad)
➡ ángulo de rotación X
EntityYaw# (entidad)
➡ ángulo de rotación Y
EntityRoll# (entidad)
➡ ángulo de rotación Z
```

Podemos encontrar en Blitz3D cuatro funciones realmente interesantes para implementarlas en comportamientos (IA) de entidades. Nos referimos a: *EntityVisible, EntityDistance, EntityPick y LinePick.*

```
EntityVisible ( entidad
fuente, entidad destino )
```

Esta función retornará "verdadero (True o 1)" si la "entidad fuente" puede ver a la "entidad destino".

```
EntityDistance# (entidad
fuente, entidad destino)
```

Con esta función obtendremos la distancia existente entre una entidad ("entidad fuente") y otra ("entidad destino").

```
EntityPick (entidad fuente,
rango#)
```

Esta función devolverá la entidad más cercana a "entidad fuente" en un radio de acción definido por "rango". Pero antes de poder aplicar esta función, es necesario que la entidad que se detecte tenga que ser "pickeable"o escogida, con la función *EntityPickMode*, la cual detallamos más adelante.

```
LinePick ( entidad fuente,
X#, Y#, Z#, DX#, DY#, DZ#
[,radio de la línea] )
```

Es una función muy parecida a la anterior, la diferencia es que ésta retornará la primera entidad que se encuentre en una línea desde las coordenadas X, Y y Z hasta X + DX, Y + DY y Z + DZ.

Para completar nuestra búsqueda de información sobre entidades encontramos dos funciones que nos dirán si una entidad está animándose y cuál es la duración de la animación:

```
EntityAnimating (entidad)
```

➡ Devuelve verdadero o falso si la animación de la entidad está activa o no.

```
EntityAnimTime# (entidad)
```

➡ Devuelve un valor numérico indicando la duración de la animación.

Para terminar, podemos transformar un punto, vector o normal de una entidad a otra con las funciones *TFormPoint, TFormVector* y *TFormNormal*. Estas funciones son muy interesantes, por ejemplo, para saber el lugar exacto del terreno donde ha caído una bomba.

Las coordenadas transformadas se pasan a las funciones *TFormedX, TFormedY* y *TFormedZ*.

```
TFormPoint X#, Y#, Z#, entidad
fuente, entidad destino.
```

Si en lugar de la entidad fuente o destino pusiéramos el valor 0, se usaría el espacio global para el cálculo.

Para terminar, al igual que podíamos obtener la madre de una entidad, con la función *GetChild* podemos obtener el hijo:

```
GetChild (entidad, índice de
la entidad hijo).
```

El "índice de la entidad" corresponde al número de la entidad que queremos. Para saber el número de entidades hijo debemos contarlas con la función *CountChildren(entidad)*.

COLISIONES ENTRE ENTIDADES

La detección de colisiones entre entidades es fundamental para interrelacionarlas y darles una presencia física total en un mundo 3D. En un videojuego, es el proceso que da vida realmente a las evoluciones de la acción. Su tratamiento puede determinar el rendimiento final de un juego, por eso es importante tener bien claro cómo implementarlas. Hay ciertos pasos que hay que seguir para implementar un sistema de colisiones en Blitz3D. Supongamos que queremos detectar la colisión entre dos objetos, por ejemplo, una esfera y un cubo que sirve de pared. En primer lugar, debemos asignarles un número para darlos a conocer al sistema de colisión. Esta operación la realizamos con la función *EntityType* de la siguiente manera:

7

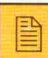
Hay tres métodos de detección de colisiones.

```
Esfera=CreateSphere():
Pared=createcube()
Const ENTIDAD_ESFERA = 1
Const ENTIDAD_PARED = 2
EntityType Esfera, ENTIDAD
_ESFERA ; ← Es igual que
EntityType Esfera,1
EntityType Cubo, ENTIDAD _PARED
```

Utilizamos constantes para facilitar luego la labor de identificar a cada entidad, importante en el caso de que tengamos una enorme lista de objetos a detectar. A continuación asignamos un número a cada objeto.

A continuación, debemos activar la colisión entre los dos tipos de entidades y decirle al Blitz3D qué acción queremos que realice cuando la colisión ocurra. Para ello utilizaremos la función:

```
Collisions entidad fuente,
entidad destino, método de
detección, tipo de acción.
```



NOTA

Varias entidades pueden tener asignada la misma variable de colisión. Por ejemplo, varias esferas diferentes pueden ser del tipo ENTIDAD_ESFERA.



NOTA

Es importante saber que el sistema no realizará ninguna acción después de una colisión hasta que no se actualice el mundo con la función *UpdateWorld()*.

El método que utiliza el sistema de colisiones para la detección de la entidad fuente con otras entidades consiste en rodearla de una esfera imaginaria, la cual servirá para detectar contactos. Es importante, por tanto, decirle al sistema de colisión del Blitz3D el tamaño que tendrá la entidad fuente (es decir, el radio de esa esfera imaginaria) para que pueda ser chequeada. Para proporcionar esta información utilizaremos la función:

```
EntityRadius entidad, radio#
```

B3D utiliza diferentes métodos de detección de colisiones (Fig. 6):

- **ESFERA a ESFERA:** método de detección = 1
- **ESFERA a POLÍGONO:** método de detección = 2
- **ESFERA a CUBO:** método de detección = 3

Cuando nos referimos, por ejemplo, al método "ESFERA a POLÍGONO", queremos decir que la entidad fuente (ESFERA) detectará cada polígono de la entidad destino (POLÍGONO). En este caso, no hace falta decirle al sistema de colisiones el tamaño de la entidad destino porque ya lo sabe.

Sin embargo, si vamos a detectar a la entidad destino con los métodos "ESFERA" y "CUBO", sí es necesario proporcionar el tamaño. Necesitaremos entonces la función:

```
EntityBox entidad, X#, Y#, Z#,
Ancho#, Alto#, Profundidad#.
```

Esta función define el tamaño de un cubo imaginario alrededor de la entidad.

Es preciso comentar que no todos los métodos de colisión funcionan de la misma forma, es decir, unos serán más precisos que otros y más lentos o rápidos. Así el método más preciso y lento es "ESFERA a ESFERA" y el menos preciso y rápido "ESFERA a CUBO".

Además de detectar colisiones, B3D puede generar un determinado tipo de acción cuando esto ocurra. Podemos elegir entre tres: parar, desplazar y caer o desplazar y seguir (Ver ejemplo8_5.bb).

Pongamos un ejemplo:

```
...
EntityRadius Bionave,50
Const ENTIDAD_BIONAVE = 1 :
Const ENTIDAD_EDIFICIO = 2
EntityType Bionave, ENTIDAD
_BIONAVE : EntityType
Edificio, ENTIDAD _EDIFICIO
Collisions Bionave, Edificio, 2, 2
If EntityCollided ( Bionave,
ENTIDAD_EDIFICIO) = True Text
10,10,"Se ha producido colisión"
...
```

En este ejemplo, le decimos al sistema que la entidad "Bionave" detecte a la entidad "Edificio" por el método "ESFERA a POLÍGONOS" y que cuando esto ocurra "Bionave" se desplazará por las paredes. A continuación, con la función:

```
EntityCollided ( Entidad,
Tipo de entidad )
```

podremos saber exactamente si ha habido una colisión específica.

Todas las colisiones ocurridas se van añadiendo a una lista denominada "Lista de información de colisiones". Esta lista es conveniente borrarla cuando no se utilice. Para ello debemos usar la función *ClearCollisions*.



NOTA

El sistema de B3D detecta siempre las colisiones que sufre la entidad fuente con la entidad destino y para ello utiliza siempre el método ESFERA a (método)



En el próximo número...

... aprenderemos a crear y manejar terrenos y nos iniciaremos en el uso de mapas BSP.

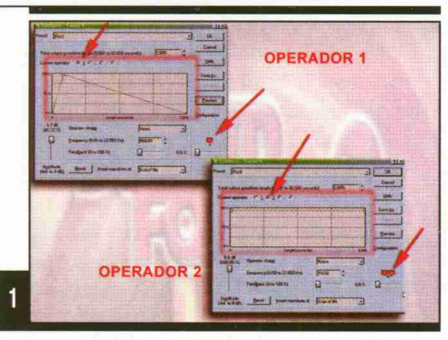
Editores de audio. Sound forge 6.0 (II)

Sound Forge es un programa muy potente con multitud de opciones nuevas y únicas para la creación y manipulación de audio. La semana anterior descubrimos cómo comenzar a trabajar con él y algunos de sus más "íntimos secretos". Cerraremos este tutorial explicando paso a paso cómo crear una fantástica explosión.

A continuación, obtendremos un bucle de sonido para utilizarlo en nuestro secuenciador favorito.

CREAR SONIDOS NUEVOS. SÍNTESIS

Sound Forge 6.0 posee una función bastante interesante que nos permite crear nuevos sonidos a partir de métodos de síntesis, similares a los que se utilizarían en un sintetizador convencional. Accedemos a ella en la opción "Tools \ Synthesis". Disponemos de tres tipos diferentes de síntesis: un generador de tonos (que genera un tono diferente a partir de números y letras introducidos en secuencia en el cuadro "Dial String), un sintetizador FM (Frecuencia Modulada, permite mezclar hasta cuatro formas de ondas diferentes con distintas configuraciones) y por síntesis simple (genera, simplemente, una forma de onda básica con tono y longitud variable). Antes de utilizar cualquiera de estos métodos es necesario crear una plantilla de audio nueva.



Esquema de las gráficas de las ondas utilizadas en la creación de la explosión.

CREACIÓN DE UNA EXPLOSIÓN

Para realizar nuestro sonido desde cero, utilizaremos el generador de síntesis FM. Creamos una plantilla nueva en "File \ New" con formato 44.100 Hz, 16 bits, estéreo. Entramos en el sintetizador FM en "Tools \ Síntesis \ FM". Vamos a utilizar dos operandos con moduladores de ruido o "Noise". Desplazamos el deslizador de configuración hacia la derecha para crear otro operando y colocarlo a continuación del primero (en serie). A continuación, dibujamos las gráficas de cada operando como se muestra en la figura 1.

Para seleccionar cada operando u onda utilizamos las casillas numeradas en "Current operator". Una vez dibujadas las gráficas de cada operando, asignaremos los valores a cada uno de ellos. Para el primero asignamos una duración de salida de 2 segundos en "Total output waveform length", un volumen (Amplitude) del 80 u 82 % y una frecuencia (Frequency) de 500,00. Para el segundo asignamos la misma duración, un volumen del 100% y una frecuencia de 10,00. Pulsando en el botón "Preview" podemos oír el resultado. Elegimos la opción "Start of file" en "Insert waveform at" para colocar el nuevo sonido al comienzo de la plantilla y pulsamos "Ok".

El siguiente paso es expandir el estéreo para lograr más sonoridad. Elegimos la opción "Process \ Pan/Expand". En la gráfica podemos modificar la expansión del sonido a través de los dos canales estéreo. Seleccionamos un volumen (Output gain) de 5.0 dB y el modo "Pan" (preserve stereo separation). La gráfica se muestra en la figura 2.



Esquema de las gráficas utilizadas en la creación de la explosión para el cambio panorámico y de tono.

Para lograr el efecto del impacto debemos modificar el tono del sonido. Para ello, nos valemos del efecto "Effects \ Pitch \ Bend". Aquí podemos dibujar a nuestro gusto la evolución del tono a lo largo del tiempo. Deslizamos la barra de semitonos "Semitones" al máximo, es decir +24, mantenemos deseleccionado la opción "Preserve original duration" y dibujamos la gráfica como se muestra en la

TRUCO

Mientras el previo está sonando podemos modificar la gráfica para oír en tiempo real los cambios.

NOTA

Para mover o crear un punto en las gráficas utilizamos el botón izquierdo del ratón y para borrarlo el derecho.

figura 2. Para conseguir un sonido más grave es imprescindible ecualizarlo. Utilizaremos para ello el ecualizador gráfico situado en "Process \ EQ \ Graphics". Podemos dibujar una gráfica de la ecualización, aunque también es posible utilizar deslizadores por cada banda seleccionando las pestañas "10 Band" o "20 Band" situadas en la parte inferior de la ventana. (Ver Fig. 3).

Antes de terminar siempre es conveniente normalizar el volumen en "Process \ Normalize". Elegimos la opción por defecto "Maximize peak value" y pulsamos "Ok".

Nuestra explosión está casi terminada. Sólo queda darle espectacularidad aplicando un poco de distorsión en "Effects \ Distortion". Situamos "Dry out" en –20 dB y "Wet out" a –3 dB. A continuación dibujamos la gráfica que se muestra en la figura 3.

Sólo nos queda ajustar la duración borrando parte del final y realizando un "Fade out" para realizar una bajada de volumen al final.

Hemos visto que a partir de una simple forma de onda podemos generar cualquier tipo de efecto especial, sólo es cuestión de saber aplicar los efectos y procesos adecuados para buscar lo que se quiere. A continuación, vamos a aprender cómo crear bucles o loops de audio a partir de un sonido.

 CREANDO LOOPS

Los loops son sonidos que, al ser reproducidos de forma infinita, parecen no tener principio ni fin. Vamos a partir del efecto de explosión que hemos creado. Hacemos una selección.

Seguidamente, para crear un loop de ahí, seleccionamos la opción "Special \ Insert Sample Loop" o pulsamos la tecla "L". Aparece la ventana "Edit Sample", la cual nos ayudará a configurar nuestro loop a medida. En nuestro caso, es suficiente con pulsar "Ok". La selección que hicimos queda incluida entre dos marcadores, que indican que el loop está creado.

En ocasiones puede ocurrir que el loop creado no comience o acabe realmente bien y en su reproducción continua se note el principio y el fin. Para solucionar este problema disponemos de la función "Crossfade Loop", que permite, por medio de "fades" entre los canales, conseguir un loop perfecto. Es importante saber que para realizar esta operación es necesario que haya espacio antes y después del loop. Elegimos la opción "Tools \ Crossfade Loop". Activamos las casillas "Loop" y "Post-Loop", situamos los deslizadores al máximo (100%) y pulsamos en "Ok". Creamos una plantilla nueva del mismo formato y copiamos el bucle (seleccionado) a ella. Pulsamos CTRL + C, hacemos clic en la nueva plantilla y pegamos con CTRL + V.

Para oír el resultado, activamos la reproducción continua pulsando en el icono correspondiente y a continuación hacemos "Play" (barra espaciadora).

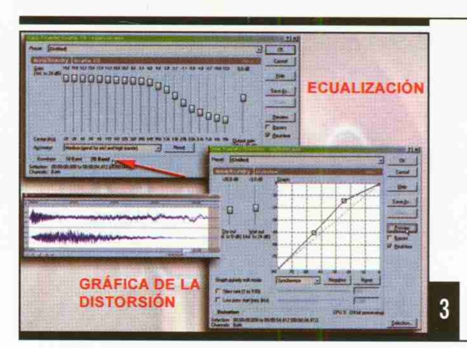
TRUCO

Podemos desplazar un loop creado a través de la línea de tiempo con las teclas " < " y " > ". Si una vez desplazado el loop queremos mantenerlo en la nueva posición, hay que elegir la opción "Update" del menú flotante que aparece al pulsar con el botón derecho del ratón sobre las marcas verdes situadas sobre los límites del loop.

 CREANDO UNA LISTA DE SELECCIONES

Sound Forge 6.0 nos permite crear una lista con diferentes selecciones de un mismo sonido o de varios. Podremos guardarla en el disco, reproducirla o utilizarla para crear nuevos sonidos.

Carguemos un sonido, por ejemplo, el de la explosión que creamos anteriormente, y hagamos cualquier selección con el ratón. Seguidamente, elegimos la opción "Special \ Region List \ Insert" para crear una lista nueva. Aparece una ventana de diálogo, donde asignamos un nombre a la selección o región (por defecto va numerado: 01, 02, etc.). Continuamos pulsando en "Ok".



Esquema de la ecualización establecida en la creación de la explosión, así como una gráfica de la distorsión aplicada.

La región queda englobada entre dos líneas de selección y con el nombre que le asignamos. Pulsamos ALT + 1 para mostrar la lista de regiones que hemos creado. En ella, podemos observar cómo contiene un elemento llamado "Selección 1". Si pulsamos en el pequeño icono de su izquierda podremos reproducir sólo esa selección. Además, podemos crear una nueva plantilla automáticamente si desplazamos cualquier elemento de la lista hacia cualquier lugar del fondo.

Hasta aquí este tutorial sobre Sound Forge 6.0. Aunque en líneas generales resulta básico es suficiente para abrir el camino a las funciones más interesantes que esta estupenda herramienta de edición de audio posee.

NOTA

Para poder crear nuevos sonidos a través de la lista de regiones es necesario que todo esté en el mismo formato de audio.



En el próximo número...

... estudiaremos las posibilidades de edición y multipista del Cool Edit Pro 2.0.

Arcades. Shoot´em up

¿Qué jugador no ha tenido entre sus manos alguna vez un matamarcianos? Siguiendo con nuestra pequeña historia de los arcades, toca ahora hablar de los más populares, los *shooters*.

✍ DESCRIPCIÓN

Este tipo de juegos ha estado siempre vinculado a las dos dimensiones y sobre todo a recreativas y consolas de principios de los 90. La estructura general de estos juegos se basa en la temática del método "dispara y olvida". El núcleo principal gira alrededor de una acción dinámica y trepidante, la cual evoluciona linealmente a través de niveles o escenarios que generalmente permanecen constantes. Suelen ser enormemente adictivos, ya que demandan toda la atención del jugador debido al corto tiempo de reacción de que éste dispone durante el juego. Los primeros matamarcianos constaban de un decorado fijo en donde evolucionaban hordas de enemigos a los cuales había que destruir, disparándoles sin tregua. Fueron este tipo de arcades los que encabezaron la historia de los videojuegos, como *Space Invaders, Galaxians o Defender.*

✍ EVOLUCIÓN TÉCNICA

La única evolución que han sufrido ha sido la mejora de gráficos, sonidos y representación en pantalla. Pero, en realidad, siempre han mantenido la misma filosofía: disparar a todo lo que se mueva. Técnicamente, estos juegos se caracterizan por poseer una gran multitud de gráficos simultáneos en pantalla.
Habitualmente, en los primeros shooters, el protagonista se encuentra en la parte inferior de la pantalla, cuyos movimientos se limitan a ir de izquierda a derecha. Su misión consiste en disparar para destruir a grupos de enemigos con movimientos prefijados y que a su vez pueden disparar también. La dificultad aumenta a medida que el jugador elimina estos grupos, apareciendo nuevos y más potentes conjuntos de enemigos con diferentes comportamientos.
Generalmente, antes de pasar de una etapa a otra, hay que destruir al jefe o enemigo final del nivel.

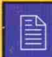
Con la llegada de las técnicas de scroll simple de pantalla y parallax, el sistema de juego se mantiene, pero la representación de la acción se realiza mediante el desplazamiento horizontal y/o vertical de cada nivel.

Aparecen entonces los decorados interactivos, los cuales se sumaban a las hordas enemigas con elementos de ataque como cañones u otros vehículos. El gé-



NOTA

Los shooters 2D han sido siempre los juegos más abundantes en los salones recreativos. Muchos de esos juegos han sido convertidos a PC o se encuentran disponibles a través de emuladores especiales.



NOTA

Quizás uno de los mejores shooters de scroll horizontal para recreativas sea la serie *R-Type* de la desarrolladora Irem. Para PC, sin lugar a dudas, encontramos la serie de juegos en scroll vertical *Xenon* de The Bitmap Brothers.



1

Imágenes de recreativas de los primeros shooters de la historia.



2

La serie R-Type constituyó el más claro ejemplo de shooter horizontal para recreativas.



3

Xenon 2000 fue el último intento de The BitMap Brothers de shooter con scroll vertical para PC.

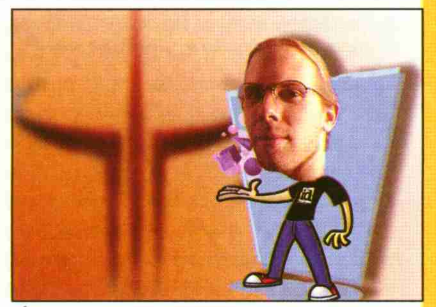


LA BIOGRAFÍA...

JOHN CARMACK

Creador del **Shooter 3D**

Junto con Adrian Carmack y John Romero fundó ID Software, compañía creadora del shooter en 3D. Está considerado como el programador de juegos y tecnología 3D en tiempo real más importante e innovador del mundo. Su filosofía es ir siempre de la mano con los nuevos avances tecnológicos en gráficos para ordenador. Gracias a esto se ha convertido en un gurú de los videojuegos con creaciones como la saga *Wolfenstein*, *Doom* y *Quake*. Es amante de los viejos juegos *arcade* como *Pacman* o *Space Invaders* y aprendió solo a programar. Vendió su primer juego cuando estaba en el instituto, un RPG llamado *Shadowforge*. En 1990 entró a trabajar en SoftDisk, donde programó la serie *Commander Keen*. Su tecnología ha sido muy solicitada y el motor del *Quake II* sirvió para desarrollar juegos como *Soldier of Fortune* o *Half Life*. Carmack es amante de la enseñanza y son conocidas sus andaduras por el mundo dando infinidad de conferencias. Una muestra de esta afición es la posibilidad de adquirir por Internet el código de sus juegos más importantes.



John Carmack.

nero aumenta de variedad, y el desarrollo de al acción transcurre en decorados muy diferentes. Ya no sólo se destruyen naves alienígenas en el espacio exterior, sino también aviones de la segunda guerra mundial, submarinos, soldados o tanques.

Una gran diferencia entre estos arcades con scroll y los de plataformas es que el escenario siempre está en movimiento, obligando al jugador a estar atento en todo momento al desarrollo del juego. A medida que la acción avanza y más enemigos son destruidos, el jugador va obteniendo incentivos, como más potencia de fuego, puntos o vida extra.

Realmente, con la llegada del 3D, la producción de shooters al estilo clásico para PC prácticamente acabó después de la serie *Xenon*. Al igual que ocurría con los juegos de plataformas, sólo se realizan *remakes* de recreativas o consolas. Poco a poco, el 3D comía terreno a los scroll de pantalla en 2D y el género iba adquiriendo otros matices, sobre todo en la jugabilidad, debido a que se añadían cada vez más tonos estratégicos a la acción. Se traslada el género y empiezan a aparecer shoot´em up subjetivos de todas las formas conocidas: combates aéreos, combates de tanques, submarinos o incluso entre soldados. Nace el *First Person Shooter* o *FPS*, encabezado por los legendarios *Doom I* y *II* y *Wolfenstein 3D*.



NOTA

Battlezone (1980) fue el primer shooter realizado en 3D. Ocurrió en las recreativas y simulaba un combate entre tanques con modelos vectoriales.



NOTA

El primer FPS con posibilidad multijugador fue realmente un juego para el Atari ST llamado MIDI Maze, el cual se unía a otro Atari a través del conector MIDI.

La tecnología 3D crece y los FPS se adueñan del mundo lúdico para PC. Tras la aparición y éxito apabullante del juego *Quake* de Id Software (1996), todas las desarrolladoras se suben al carro de este género. Con la llegada de la técnica multijugador con el juego *Doom*, los FPS se transportan a un nivel insospechado de jugabilidad. Ahora, los alienígenas pueden ser creados por el ordenador o ser otros jugadores. Pero a pesar de toda esta tecnología, el argumento de un Shoot´em up mantiene sus cualidades: disparar a todo lo que se mueva.

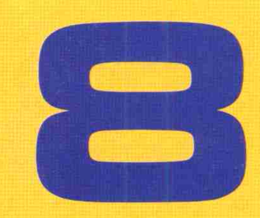
La perfección de los sistemas multijugador propició la aparición de un nuevo tipo de FPS, los *arena*. En ellos, jugadores conectados a la misma partida luchan entre sí en niveles de pequeña extensión. Destacamos verdaderos juegos de culto como *Unreal Tournament* (Epic Games) o *Quake III arena* (Id Software). El motor gráfico de éste último ha sido muy solicitado para el desarrollo de otros títulos.

Cada vez más, el protagonista adquiere forma humana, dejando atrás naves espaciales o tanques. El género FPS a veces se transforma y entonces la cámara subjetiva se eleva, mostrando al personaje por completo. Aparecen entonces los primeros juegos en tercera persona, que en cierta manera recuerdan a los clásicos shooters de scroll de pantalla. Poco a poco, se irán mezclando factores estratégicos y lógicos en el desarrollo del juego. La incesante acción se vuelve aventura y completar un nivel ya no se reduce a destruir al malo más grande. Sin duda, el juego que marcó un antes y un después en la acción 3D fue *Tomb Raider*, que introdujo el concepto de aventura en los shoot´em up 3D para PC.

En el próximo número...

... hablaremos con más profundidad de este magnífico juego y de los demás arcades 3D que siguieron sus pasos y crearon diferentes subgéneros dentro de los FPS.

Cuestionario Videojuegos



Preguntas

1. ¿Qué funciones nos proporciona Blitz3D para mover y rotar una entidad?
2. Describe el procedimiento necesario para que Blitz3D pueda detectar la colisión entre dos entidades.
3. ¿Qué ventajas se obtienen de dividir el código en distintos módulos?
4. Al utilizar un sistema modular de programación debemos definir un número mayor de variables globales, ¿por qué?
5. Antes de poder pintar sobre el modelo 3D en Deep Paint 3D, ¿qué proceso debemos realizar?
6. ¿Cómo podemos pintar con más detalle una pieza del modelo en Deep Paint 3D?
7. ¿Qué proceso debemos realizar en GoldWave si queremos que un sonido suene más rápido?
8. ¿Qué procedimiento debemos realizar para que un sonido termine con una bajada de volumen?
9. ¿Cómo podemos crear un loop perfectamente continuo con cualquier selección en Sound Forge 6.0?
10. ¿Cómo podemos crear en Sound Forge 6.0 una lista de regiones?

Respuestas al cuestionario 7

▷ **1.**
```
GetColor(X,Y)
Rojo=ColoRed(): Verde=ColorGreen(): Azul=ColorBlue()
; Cambiar valor de Rojo, Verde y Azul
Color Rojo, Verde, Azul
```

▷ **2.** SpriteViewMode Sprite, modo de orientación
Modo 1: El sprite siempre mira a la cámara.
Modo 2: El sprite queda libre de orientación.
Modo 3: Se orienta a la cámara pero no cuando ésta se inclina.
Modo 4: Orientación completa, incluyendo giros e inclinaciones.

▷ **3.**
```
mx#=mx# - Float (MouseYSpeed()) / -7
mx=mx / 1.5
my#=my# + Float (MouseXSpeed()) / -7
my=my / 1.5

X# = EntityPitch ( camara ) + mx
Y# = EntityYaw   ( camara ) + my
```

▷ **4.** Las funciones:
```
CameraPick (camara, MouseX(), MouseY())
Y    PickedX()   PickedZ()
```

▷ **5.** Seleccionando y uniendo sus vértices con la función "Snap".

▷ **6.** En .3DS o .OBJ.

▷ **7.** Para cambiar la frecuencia de muestreo de un sonido.

▷ **8.** Modificando el panorámico por medio de la función "Pan".

▷ **9.** Seleccionando el modo "Create a new window for each take".

▷ **10.** Utilizando la síntesis FM en el menú de herramientas "Tools".

Contenido CD-ROM 8

▶ AUDIO

■ Virtuosa Phoenix Edition 4.01



Excelente reproductor que soporta además múltiples formatos de ficheros de audio.

■ Power Audio Editor 3.0.5

Podremos reproducir, crear, editar y guardar ficheros de audio en diversos formatos y aplicando múltiples efectos.

■ Storm 5.1

Completa herramienta de creación de música, que incluye infinidad de funciones, instrumentos y opciones.

■ MIDINight Express 2.7

Potente reproductor MIDI que emula la reproducción por tabla de ondas.

■ Res Rocket 1.4

Con esta simpática aplicación podrás crear música a través de internet con quien tú quieras.

■ Goldwave 4.6

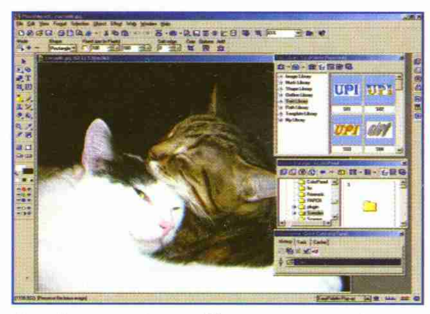
Nueva oportunidad para hacerte con este excelente programa.

▶ DISEÑO 2D

■ Adesign 1.0

Software para diseñar imágenes en dos dimensiones que además incorpora herramientas para bitmaps y vectores.

■ Ulead PhotoImpact 8.0



Crea logos y otros gráficos con este software que soporta también imágenes en 3D.

■ Image Broadway 4.1

Editor de imágenes que además incorpora un corrector de ojos rojos.

■ Autoimager 2.3

Podrás procesar y convertir imágenes en más de 70 formatos diferentes.

■ CeledyDraw 1.5

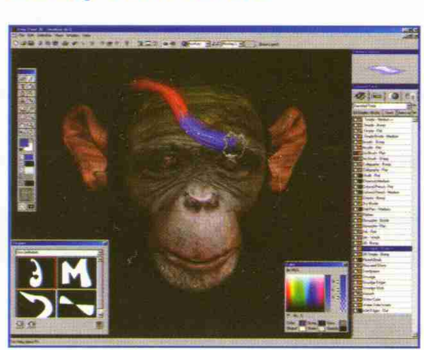
Interesante software con el que podrás hacer auténticas virguerías.

■ Pixopedia 24 0.6

Añádele a tus imágenes textos, contornos, formas o movimiento.

▶ DISEÑO 3D

■ Deep Paint 3D 2.0



Demo del popular programa en esta versión que soporta *MercatorUV*.

■ Beyond 3D Extreme 2.01 Beta

Crea mundos virtuales aplicando además texturas y otros efectos.

■ Genesis 3D Editor 1.5

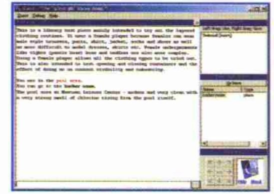
Crea objetos 3D renderizados en tiempo real.

■ Softy 3D 1.0

Aplicación idónea para crear complejos de tipo orgánico de un modo muy sencillo.

▶ PROGRAMACIÓN

■ Quest 3.11



Crea tus propias aventuras conversacionales con la ayuda de esta aplicación.

■ Morfit 3D y Euphoria

Euphoria es un lenguaje de programación sencillo y potente que, con la ayuda de Morfit 3D, nos permitirá programar espectaculares juegos en poco tiempo.

■ Dings Game Basic

Lenguaje de programación diseñado especialmente para poder programar juegos rápidamente.

■ SCI Studio

Software imprescindible y muy potente para desarrollar juegos en tres dimensiones.

▶ JUEGOS

■ Battlezone

Primer *shooter* en 3D de la historia. Deberás luchar por tener el control de la tecnología alienígena.

■ Doom 1.9

Mata todos los demonios que puedas usando tu arsenal de armas.

■ Unreal Tournament

Magnífico juego con el que podrás competir en red contra todos tus amigos.

■ Wonfenstein 3D

Legendario juego en el que deberás huir de una prisión nazi.



■ Zone of Fighters

Nueva versión de nuestro juego, con muchas más mejoras.

▶ VÍDEO

■ Capture Studio Professional 4.01

Con este útil programa podrás capturar tanto vídeo como imágenes.

■ FadeToBlack 2.0.5

Herramienta que te será de gran ayuda para editar vídeo y ficheros AVI.

■ Full Motion video Pro 4.8



Este editor de vídeo incorpora además múltiples efectos.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.